TEKNILLINEN KORKEAKOULU
Automaatio- ja systeemitekniikan osasto

Ville Hietanen

Klusterointimenetelmien hyödyntäminen yksikköprosessien suorituskyvyn seurannassa

Valvoja: Professori Heikki Koivo, Systeemitekniikan Laboratorio, TKK
Ohjaaja: TkT Mats Friman, Metso Automation OY

Tekijä: Ville Hietanen
Osasto: Automaatio- ja systeemitekniikan osasto
Pääaine: Systeemitekniikka
Sivuaine: Automaatiotekniikka

Työn nimi: Klusterointimenetelmien hyödyntäminen yksikköprosessien suorituskyvyn seurannassa

Title in English: Use of Clustering Methods in Unit Process Performance Monitoring

Professuurin koodi ja nimi: AS-74 Systeemitekniikka

Työn valvoja: Professori Heikki Koivo

Työn ohjaaja: TkT Mats Friman

Tiivistelmä:

Teollisten järjestelmien monitorointiin ja diagnostiikkaan on kehitetty useita eri menetelmiin perustuvia sovelluksia. Kehitetyistä sovelluksista on saatu hyviä kokemuksia, mutta niiden on havaittu antavan tietyissä ajotilanteissa harhaanjohtavaa informaatiota kohdejärjestelmän tilasta.

Eri ajotilanteet pitäisikin ottaa huomioon monitorointisovelluksessa. Käyttötilanne saattaa riippua useista muuttujista ja tilatiedoista, jolloin niiden konfigurointi muodostuu ongelmaksi.

Tässä työssä esitellään klusterianalyysiin perustuva ratkaisu eri ajotilanteiden konfiguroinnin helpottamiseksi. Klusterianalyysin avulla voidaan toteuttaa yleiskäyttöinen menetelmä käyttötilanteiden oppimiseksi ja tunnistamiseksi mittauksista, jolloin tämä voidaan huomioida monitorointi- ja diagnostiikkasovelluksissa.

Klusterointimenetelmät kykenevät myös mallintamaan analysoitavassa datassa olevia riippuvuuksia, jolloin klusterianalyysin avulla voidaan luoda staattisia malleja monimutkaisille järjestelmille. Rekursiivisella klusterointialgoritmilla mallia voidaan päivittää kaiken aikaa uusien mittausten perusteella.

Itseorganisoituvaa karttaa on sovellettu samankaltaisiin ongelmiin. Itseorganisoituvan kartan opetusalgoritmi on kuitenkin naapuruussuhteineen monimutkainen toteuttaa. Mallinnussovelluksessa koodikirjan järjestyksellä ei kuitenkaan ole merkitystä, jolloin mallin muodostukseen voidaan käyttää huomattavasti yksinkertaisempia klusterointialgoritmeja.

Sivumäärä: 85+2

Avainsanat: Klusterointi, Itseorganisoituminen, Ennakoiva kunnossapito, Ajotilanne

Täytetään osastolla

Hyväksytty

Kirjasto

SISÄLLYSLUETTELO

SISÄLLYSLUETTELO 3

LIITELUETTELO 5

KÄYTETTYJÄ MERKINTÖJÄ 5

LYHENTEITÄ 6

ALKULAUSE 7

1 JOHDANTO 8

1.1 YLEISTÄ MONITOROINNISTA JA DIAGNOSTIIKASTA 8
1.2 TYÖN TAVOITTEET JA RAKENNE 9

2 KUNNOSSAPIDON TUKIJÄRJESTELMIÄ 10

2.1 YLEISTÄ 10
2.2 KUNNONVALVONTA LAITETASOLLA 11
2.2.1 Säätöventtiilien kunnonvalvonta 11
2.2.2 Pyörivien laitteiden kunnonvalvonta 12
2.3 SUORITUSKYVYN SEURANTA SÄÄTÖPIIRITASOLLA 13
2.4 SUORITUSKYVYN SEURANTA YKSIKKÖPROSESSITASOLLA 14
2.5 KOKEMUKSIA KUNNOSSAPIDON TUKIJÄRJESTELMISTÄ 15

3 KLUSTEROINTIMENETELMIÄ 17

3.1 YLEISTÄ KLUSTERIANALYYSISTÄ 17
3.2 DATAN ESIKÄSITTELY 18
3.3 KLUSTEROINTIALGORITMEJA 20
3.3.1 K-means 20
3.3.2 Itseorganisoituva kartta (SOM) 21
3.3.3 Neural gas 22
3.3.4 Expectation Maximization (EM) 23
3.3.5 Fuzzy c-means 24
3.4 ALGORITMIEN OMINAISUUKSIA 26

4 AJOTILANTEEN TUNNISTUSJÄRJESTELMÄ 33

4.1 YLEISTÄ AJOTILANTEEN TUNNISTAMISESTA 33
4.2 TOTEUTETUN OHJELMISTON RAKENNE 35
4.3 KLUSTEROINTILOHKON TOIMINTAPERIAATE 35
4.4 KLUSTEROINTIMENETELMIEN SOVELTUVUUS TIETOKANTAYMPÄRISTÖÖN 37
4.4.1 Käytetyn algoritmin matemaattinen kuvaus 38
4.4.2 Algoritmin ominaisuuksia 39
4.5 SIGNAALIEN ESIKÄSITTELY 42

4.6 TOTEUTUS TIETOKANTAYMPÄRISTÖÖN 43
4.7 KONFIGURAATIOTIETOKANTA 45
4.8 OPEN DATABASE CONNECTIVITY (ODBC) 47
4.9 KÄYTTÖÖNOTTO JA MATLAB-KONFIGURAATTORI 49
4.9.1 ODBC-tietokantarajapinta Matlabissa 49
4.9.2 Tunnistusjärjestelmän opetus 50
4.9.3 Konfiguraation tallennus konfiguraatiotietokantaan 50
4.10 LIITYNTÄ MUIHIN SUORITUSKYKYÄ SEURAAVIIN SOVELLUKSIIN 51

5 SOVELLUSKOHTEITA 53

5.1 ADAPTIIVINEN SÄÄTÖ 53
5.2 DATAN PAKKAUS 54
5.3 VIKADIAGNOSTIIKKA 57
5.4 CASE: TENNESSEE EASTMAN PROSESSI 57

6 TESTAUS 63

6.1 YLEISTÄ 63
6.2 LASKENTALOHKON TESTAUS 63
6.3 LBHISTORY 66
6.4 AJETTAVAN PAPERIN LAJITYYPIN TUNNISTUS 67
6.5 TUNNISTUSJÄRJESTELMÄN KÄYTTÖÖNOTTO 70
6.6 AJOTILANTEEN MERKITYS 71

7 JATKOKEHITYS 74

7.1 AJATUKSIA KENTTÄLAITEDIAGNOSTIIKASTA 74
7.2 AJATUKSIA YKSIKKÖPROSESSIEN MONITOROINNISTA 76

8 YHTEENVETO JA JOHTOPÄÄTÖKSET 79

LÄHDELUETTELO 81

Liiteluettelo

Liite 1 Työssä kehitetyn ohjelmistopaketin rakennekaavio
Liite 2 Matlab funktiot

Käytettyjä merkintöjä

$\#\{\Gamma\}$	Näytteiden lukumäärä klusterissa Γ
a	Unohduskerroin
$d(\boldsymbol{x}_1,\boldsymbol{x}_2)$	Vektoreiden $\boldsymbol{x}_1$ ja $\boldsymbol{x}_2$ välinen etäisyys
$\det\{\boldsymbol{M}\}$	Matriisin $\boldsymbol{M}$ determinantti
E	Kvantisointivirhe
Γ	Klusteri, Neuroni
h	Koodivektorin päivityksessä käytettävä parametri
k	Koodivektorien lukumäärä
$\boldsymbol{M}^T$	Matriisin $\boldsymbol{M}$ transpoosi
μ_x, $\bar{\boldsymbol{x}}$	Näytteiden aritmeettinen keskiarvo (1 x p)
n	Näytteiden lukumäärä
p	Näytevektorin dimensio
$p(\boldsymbol{x})$	Näytteen $\boldsymbol{x}$ todennäköisyys
R^2	Selitysaste
σ_x^2	Näytteiden varianssi (1 x p)
τ	Dynaamisen järjestelmän aikavakio
t	Aikaindeksi
$u(t)$	Systeemin tulomuuttuja
$\boldsymbol{v}^j$	Koodikirjan j:s koodivektori (1 x p)
$\boldsymbol{V}$	Koodivektoreiden muodostama koodikirja (k x p)
w	Painokerroin
$\boldsymbol{X}$	Näytejoukko (n x p)
$\boldsymbol{x}(i)$	Näyte hetkellä i. Näytejoukon i:s rivi (1 x p)
$\boldsymbol{x}_i$	Näytejoukon $\boldsymbol{X}$ i:s sarake (n x 1)
$y(t)$	Systeemin lähtömuuttuja

Lyhenteitä

APROS	Advanced Process Simulator
API	Application Protocol Interface
ARX	Auto-Regressive eXogenous
CBR	Case Based Reasoning
CON	CONtrol signal
DCS	Distributed Control System
DNA	Dynamic Network of Applications
DTM	Device Type Manager
EM	Expectation Maximization
FCM	Fuzzy C-Means
FDT	Field Device Tool
HALOT	Hierarkinen Analysointijärjestelmä tuotantoLinjan Osaprosessien tilan Tarkkailuun
MLP	Multi Layer Perceptron
MSPC	Multivariate Statistical Process Control
ODBC	Open DataBase Connectivity
OVI	Osaprosessien VIkadiagnostiikka
PCA	Principal Component Analysis
PID	Proportional, Integral, Derivative
PLS	Projection to Latent Structures
PV	Process Value
SC	Super Calandered
SOM	Self-Organizing Map
SP	SetPoint
SPC	Statistical Process Control
SQL	Structured Query Language
STD	STandard Deviation
TMP	Thermo Mechanical Pulp
XML	eXtensible Markup Language

Alkulause

Tämä työ on tehty Metso Automation OY:n tutkimus- ja tuotekehitysyksikössä vuosien 2002 ja 2003 aikana. Työ liittyy tutkimusprojektiin, jonka tarkoituksena oli kehittää yleiskäyttöisiä menetelmiä kenttälaitteiden ja yksikköprosessin suorituskyvyn määrittämiseksi.

Haluan kiittää työn valvojaa professori Heikki Koivoa, Teknillisen korkeakoulun systeemitekniikan laboratorion henkilökuntaa ja työn ohjaajaa TkT Mats Frimania mielenkiintoisesta aiheesta ja heiltä saamistani neuvoista. Lisäksi haluan kiittää työkavereitani Metso Automation OY:n tuotekehitysyksikössä hyvistä neuvoista, kommenteista ja hyvästä työilmapiiristä sekä DI Riku Lähdemäkeä UPM-Kymmene OYJ:stä, joka auttoi kehitettyjen ohjelmistojen testaamisessa.

Haluan kiittää myös perheenjäseniä, jotka ovat osaltaan tukeneet opiskeluani Teknillisessä korkeakoulussa, sekä opiskelukavereita joiden ansiosta opiskelu Teknillisessä korkeakoulussa on ollut kaikinpuolin positiivinen kokemus.

Otaniemessä 04.03.2003

Ville Hietanen

1 Johdanto

1.1 Yleistä monitoroinnista ja diagnostiikasta

Kiristynyt markkinatilanne ja kaiken aikaa tiukentuvat päästörajoitukset vaativat prosessiteollisuudelta tehokkaampia valmistusprosesseja, tasalaatuisempia lopputuotteita sekä tehokkaampaa raaka-aineiden ja energian käyttöä.

Kilpailu on kiristynyt myös automaatiotoimittajien kesken. Automaatiovalmistajat haluavat ottaa suuremman vastuun toimittamistaan järjestelmistä. Tällainen liiketoiminta vaatii pitkälle jalostettua informaatiota prosessilaitteiston kunnosta ja suorituskyvystä.

Nykyisin prosesseista saatavan prosessidatan määrä on valtava ja sitä varastoidaan massiivisiin tietokantoihin. Tulevaisuudessa älykkäät kenttälaitteet lisäävät datan määrää entisestään. Vaikka dataa on paljon sen jalostaminen hyödylliseksi informaatioksi vaatii kehittyneitä menetelmiä ja työkaluja. Tällaisia työkaluja kunnossapidon tueksi onkin jo markkinoilla.

Yleiskäyttöisten työkalujen kehitys on osoittautunut erittäin haasteelliseksi. Prosessista kerätään historiatietokantaan tietoa sadoista muuttujista, joiden välillä on monimutkaisia riippuvuuksia. Prosesseissa ilmenee muutoksia, erilaisia ajotapoja, ajotilanteita, häiriöitä ja vikoja. Lisäksi kaikki prosessikokonaisuudet ovat keskenään erilaisia. Myös hyvän suorituskyvyn kriteereiden määrittely on vaikeaa. Työkalujen pitäisi kuitenkin olla helppokäyttöisiä, luotettavia ja helposti asennettavia.

Ongelmaa on yritetty ratkaista erityyppisillä menetelmillä. Nämä voidaan jakaa datalähtöisiin, analyyttisiin ja tietämyspohjaisiin menetelmiin [Chi01]. Kaikissa menetelmätyypeissä on kuitenkin käyttöä rajoittavia ongelmia.

Datalähtöiset menetelmät tekevät tilastollisia johtopäätöksiä datan perusteella. Muutos prosessin käyttäytymisessä havaitaan jonkin tilastollisen tunnusluvun muutoksena. Yksittäisten laatusuureiden monitorointi (Statistical Process Control) on helppoa, mutta aiheuttaa paljon vääriä hälytyksiä, koska muuttujien välisiä riippuvuuksia ei huomioida. Ratkaisua tähän on haettu monimuuttuja SPC-menetelmistä, joissa otetaan huomioon moniulotteisessa datassa esiintyvät korrelaatiot projisoimalla data alempidimensioiseen avaruuteen [Kou02, Mar02, Hyö01]. MSPC-menetelmät sisältävät raskaita matriisioperaatioita, mikä rajoittaa niiden käyttöä.

Analyyttisissä menetelmissä monitoroitavasta prosessista luodaan prosessimalli, johon prosessin käyttäytymistä verrataan. Mahdollinen vikaantuminen näkyy mallin ennustamien ja mitattujen arvojen eron eli mallivirheen kasvuna. Ongelmana tässä lähestymistavassa on prosessimallin herkkyys prosessimuutoksille ja eri ajotilanteiden aiheuttamalle epälineaarisuudelle. Mallinnus on työlästä ja vaikeaa, koska mallinnukseen tarvitaan prosessikokeita ja mallinnettavan prosessin perusteellista tuntemusta.

Tietämyspohjaiset menetelmät vaativat toimiakseen asiantuntijan laatiman sääntökannan. Sääntökanta koostuu yleensä jos-niin-tyyppisistä säännöistä, joilla prosessin tilaa voidaan diagnosoida. Sääntökannan luominen vie aikaa ja olemassa olevan kannan ylläpito on vaikeaa, koska sääntöjen lukumäärä voi kasvaa huomattavan suureksi.

Itseorganisoituvalla piirrekartalla [Koh95] on saatu mielenkiintoisia tuloksia, koska sen avulla voidaan luoda koodivektoreihin perustuva staattinen malli analysoitavasta järjestelmästä. Kvantisointivirheen ja mallin avulla voidaan sekä havaita vika, että diagnosoida se [Kas92, Rin00, Hol96]. Kartan opetuksessa tarvitaan paljon hyvälaatuista dataa.

1.2 Työn tavoitteet ja rakenne

Tämän diplomityön tavoitteena on:

- Kehittää yleiskäyttöinen menetelmä, jota voidaan hyödyntää yksikköprosessien suorituskyvyn seurannassa.
- Yhdistää suorituskykyanalyysistä ja diagnostiikasta saadut tulokset prosessin ajotilanne- ja ajotapatietoon.
- Toteuttaa tarvittavat laskennat DNAhistorian-tietokantaympäristöön ja testata niiden toimivuus.

Diplomityö on jäsennelty siten, että luvussa kaksi esitellään joitakin olemassa olevia kunnossapidon tukijärjestelmiä. Luvussa kolme kerrotaan klusterianalyysistä yleensä sekä esitellään tunnettuja klusterointialgoritmeja ja niiden ominaisuuksia.

Luvussa neljä esitellään työssä toteutettu järjestelmä eri ajotilanteiden tunnistamiseksi. Luvussa viisi esitellään klusterointimenetelmien mahdollisia sovelluksia. Luvussa kuusi kerrotaan työssä kehitettyjen ohjelmistojen testauksesta eräällä paperikoneella. Luvussa 7 esitetään jatkokehitysideoita ylemmän tason kunnonvalvontajärjestelmien toteuttamiseksi. Johtopäätökset ja yhteenveto esitetään luvussa 8.

2 Kunnossapidon tukijärjestelmiä

2.1 Yleistä

Tuotantoprosessi voidaan jakaa hierarkisesti eri tasoihin esimerkiksi seuraavasti [Kiv02]:

- **Prosessi**: Koko tuotantolaitos
- **Yksikköprosessi**: Tuotantolaitoksen rajattu toiminnallinen kokonaisuus, jonka toimintaa voidaan tarkastella tiettyjen mitattavissa olevien tulo- ja lähtösuureiden avulla
- **Toiminto**: Säätö- ja ohjauspiirit ympäristöineen
- **Laite**: Yksittäiset laitteet, instrumentit ...

Tuotantoprosessin eri hierarkiatasoille on kehitetty erilaisia kunnossapidon tukijärjestelmiä vikadiagnostiikkaan, kunnonvalvontaan ja suorituskyvyn seurantaan.

Järjestelmien pääasiallinen tarkoitus on parantaa tuotantolaitteiston käyttövarmuutta. Kehittyvät viat ja muut ongelmat pyritään havaitsemaan mahdollisimman aikaisessa vaiheessa, jolloin parhaassa tapauksessa vältytään tuotannonmenetyksiltä. Vikadiagnostiikkajärjestelmien tarkoitus on nopeuttaa havaittujen vikojen korjaamista. Järjestelmillä pyritään myös parantamaan prosessin tehokkuutta ja suorituskykyä eli tuottamaan tasalaatuisempia tuotteita, tehokkaammin, taloudellisemmin ja ympäristöä säästäen.

Tietoliikenneyhteydet ovat olennainen osa järjestelmiä, koska niiden avulla prosesseja voidaan monitoroida ja diagnosoida maailmanlaajuisesti. Ongelmiin saadaan ratkaisu nopeammin, kun asiantuntijan ei tarvitse matkustaa paikan päälle.

Näille järjestelmille on asetettu suuria odotuksia ja nykyään useimmat automaatio- ja laitetoimittajat tarjoavat jonkinlaisia asiaan liittyviä sovelluksia omiin järjestelmiinsä ja laitteisiinsa.

Järjestelmät mahdollistavat parhaimmillaan siirtymisen määräaikaisesta huollosta oikea-aikaiseen huoltotoimintaan, mikä tarkoittaa selvää kustannussäästöä tuotantolaitoksen omistajalle. Järjestelmät tuottavat informaatiota laitteiden kunnosta, joten järjestelmien avulla laitetoimittajan on mahdollista ennustaa laitteiden myyntiä ja sitä kautta tehostaa omia valmistusprosessejaan ja laitteiden jakelua maailmanlaajuisesti. Toimivista järjestelmistä olisikin hyötyä sekä tuotantolaitokselle että tuotantolaitoksen automaatio- ja laitetoimittajalle.

2.2 Kunnonvalvonta laitetasolla

Kenttälaitteella tarkoitetaan tuotantoprosessissa toimivaa kokonaisuudeksi käsitettävää laitetta, joka suorittaa prosessissa sille määrättyä tehtävää. Esimerkkejä kenttälaitteista ovat sähkökäytöt, pumput, puhaltimet, säätöventtiit, lämmönvaihtimet ja mitta-anturit. Laitteiksi luokitellaan myös paperikoneen lajittimet, jauhimet ja voimalaitosten höyryturbiinit, vaikka ne ovat jo huomattavasti monimutkaisempia.

Kunnonvalvontajärjestelmien näkökulmasta kenttälaitteet voidaan jakaa älykkäisiin ja perinteisiin laitteisiin. Älykkäät laitteet sisältävät mikroprosessorin ja muistia, joten niihin on mahdollista ohjelmoida erilaisia toimintoja. Älykkäät laitteet kykenevät kaksisuuntaiseen kommunikointiin digitaalisen kenttäväylän avulla. Ne pystyvät siis valvomaan omaa toimintaansa ja kertomaan mahdollisesta viasta tai huoltotarpeesta [VTT02].

Suuri osa tuotantoprosesseissa olevista kenttälaitteista on kuitenkin perinteisiä. Perinteiset laitteet eivät kykene kaksisuuntaiseen kommunikointiin, joten niiden monitorointi ja diagnostiikka täytyy toteuttaa erillisillä laitteen ulkopuolisilla järjestelmillä. Kunnonvalvontajärjestelmiä on kehitetty erilaisille pyöriville koneille, joiden kuluminen havaitaan värähtelyanalyysin avulla [Par01].

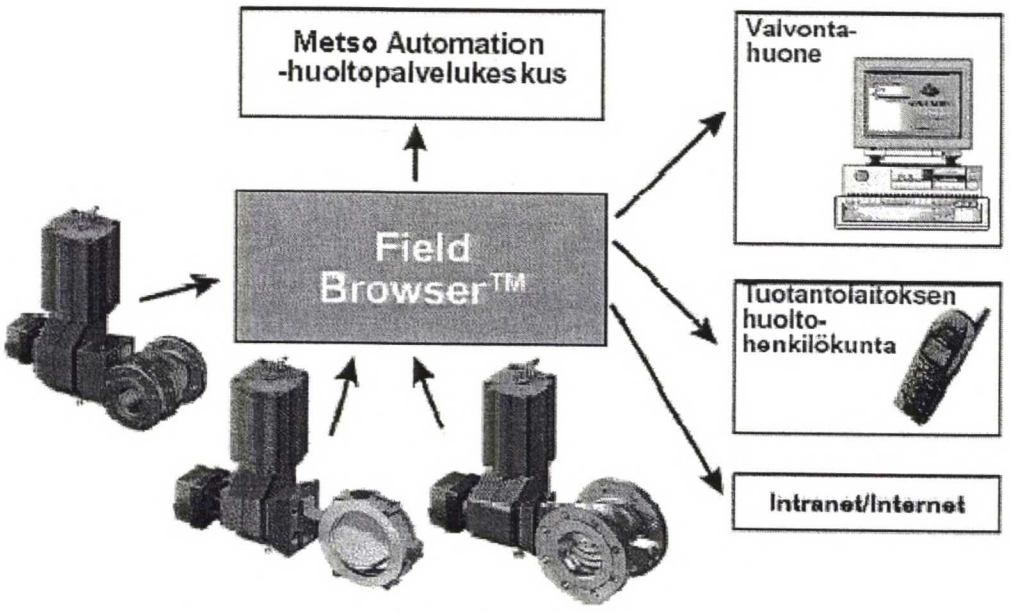
Yleisesti kunnonvalvontajärjestelmät perustuvat laitekohtaisiin tunnuslukuihin, joiden muutoksia ja kehitystä seurataan. Tunnusluvulle on määrätty raja-arvo, jonka ylittyminen tulkitaan viaksi tai huoltotarpeeksi.

2.2.1 Säätöventtiilien kunnonvalvonta

Säätöventtiilit ovat tärkeä yksittäinen prosessilaiteryhmä ja säätöventtiilien moitteeton toiminta on tuotantoprosessille tärkeää. Venttiilit ovat erityisen sopiva kohde kunnonvalvontajärjestelmälle, koska niitä on paljon, ne ostetaan yleensä samalta toimittajalta ja erilaisten venttiilien kuntoa voidaan valvoa samoilla menetelmillä.

Neles-säätöventtiileissä on pitkälle kehitetty kunnonvalvontajärjestelmä, joka perustuu venttiilin asennoittimeen integroituun diagnostiikkaohjelmistoon. Venttiilistä mitataan sen asennon ja asennon asetusarvon poikkeamaa "Travel Deviation" ja pienen liikkeen aikaansaamiseksi tarvittavaa voimaa "Actuator Load Factor". Näiden tunnuslukujen perusteella venttiilin suorituskykyä voidaan arvioida reaaliajassa prosessia häiritsemättä [Str01].

Tuotantolaitoksen venttiilien kuntoa voidaan seurata Metso Automationin kehittämällä FieldBrowser-järjestelmällä (kuva 1). Kunnossapito-organisaatio tai laitetoimittaja saa tiedon vikaantuneista ja huoltoa tarvitsevista venttiileistä automaattisesti lähetetyn sähköpostin tai tekstiviestin avulla. Huoltotoiminta nopeutuu kun huoltoa tarvitsevat venttiilit tiedetään ja kunnossapidon rajalliset resurssit kohdistuvat oikein.



Kuva 1 FieldBrowser ohjelmistolla voidaan seurata säätöventtiilien tilaa jatkuvasti [Met01A].

2.2.2 Pyörivien laitteiden kunnonvalvonta

Ensimmäiset kunnonvalvontajärjestelmät on kehitetty pyöriviin laitteisiin kuten sähkömoottoreihin, vaihteistoihin, akseleihin ja paperikoneen teloihin. Pyörivissä laitteissa kuluminen aiheuttaa värähtelyjä epänormaaleilla taajuuksilla. Pyörivien laitteiden värähtelyanalyysiin perustuvia kunnonvalvontajärjestelmiä on markkinoilla paljon.

Myös Metso Automation on kehittänyt yleisen pyörivien laitteiden kunnonvalvontaan soveltuvan Sensodec6S järjestelmän, jolla saadaan värähtelyanalyysiin perustuvaa informaatiota laitteiden kunnosta. Sensodec ohjelmistolla on analysoitu onnistuneesti paperikoneen teloja sekä höyry- ja kaasuturbiineja [Met02d].

Sähkömoottorit ovat yleisiä prosessilaitteita, joiden kunnonvalvonnan merkitys on havaittu aikaisessa vaiheessa. Sähkömoottoreissa ilmenee mekaanisten laakerivikojen lisäksi sähköisiä vikoja, jotka vaativat laitekohtaista analyysiä. Sähkömoottoreiden vikaantumisista vähän yli puolet on mekaanisia vaurioita ja loput sähköisiä [Par01,Tun98].

Sähkömoottoreiden kunnonvalvontaan on kehitetty myös palveluja. Esimerkiksi ABB tarjoaa MachineCondition Analysis-palvelua, jossa analysoitavaan moottoriin asennetaan ABB:n kehittämä ConditionAnalyzer-järjestelmä, joka sitten kytketään maailmanlaajuiseen palvelukeskukseen. Moottorin kunnosta raportoidaan määrävälein

asianosaisille [ABB02].

Värähtelyanalyysin kaltaiset yleiset kunnonvalvontaan sopivat menetelmät olisivat hyödyllisiä perinteisten laitteiden seurannassa mutta ne ovat osoittautuneet erittäin haasteellisiksi kehittää.

2.3 Suorituskyvyn seuranta säätöpiiritasolla

Säätöpiirin suorituskyvyllä tarkoitetaan säätöpiirin kykyä pitää prosessin mittaus säätöpiirille annetussa asetusarvossa häiriöistä ja asetusarvomuutoksista huolimatta mahdollisimman pienellä toimilaitteen rasituksella. Säätöpiirejä voidaan analysoida askelkokeesta laskettavilla integraalikriteereillä, mallipohjaisilla taajuustason tunnusluvuilla tai mittauksen ja asetusarvon erotuksesta eli erosuuresta laskettavilla tilastollisilla tunnusluvuilla [Har97].

Säätöpiirin suorituskyky riippuu toimilaitteen kunnosta, mittausanturin toimivuudesta, ulkoisista häiriöistä ja säätimen virityksestä. Lisäksi mittausanturin on tuotettava luotettavaa informaatiota säädettävästä prosessista ja toimilaitteen on pystyttävä vaikuttamaan säädettävään prosessiin. Yleinen syy säätöpiirin suorituskyvyn heikkenemiselle onkin mittausanturin tai toimilaitteen kuluminen tai likaantuminen, joiden vaikutuksesta säätöpiiriin syntyy kitkaa ja välystä.

Säätöpiirien huono suorituskyky aiheuttaa pahimmassa tapauksessa prosessiin ongelmia, jotka vaikuttavat raaka-aineen kulutukseen ja lopputuotteen laatuun. Esimerkiksi liian nopeasta virityksestä johtuva yksittäisen piirin värähtely heijastuu muihinkin piireihin, jolloin toimilaitteet tekevät turhaa työtä ja kuluvat nopeammin kun taas liian hitaaksi viritetyt säätöpiirit eivät kykene kompensoimaan häiriöitä ja aiheuttavat muutostilanteissa hitautta koko prosessiin.

Prosessin muuttuvat olosuhteet aiheuttavat epälineaarisuutta, mikä vaikeuttaa säätimien viritystä. PID-säätimen viritys on hyvä tietyssä toimintapisteessä, mutta prosessin toimiessa useissa toimintapisteissä ei kyseisillä parametreilla ole mahdollista hallita prosessia optimaalisesti.

Säätöpiirien suorituskykyyn on alettu kiinnittää entistä enemmän huomiota. Nykyään lähes kaikilla automaatiojärjestelmätoimittajilla onkin oma asiaan liittyvä sovellus tai palvelu, joka käsittää suorituskyvynseurannan ja virityksen.

Metso Automationin tuote on LoopBrowser [Met02b], jonka avulla saadaan informaatiota säätöpiirien suorituskyvystä. LoopBrowser-sovelluksessa säätöpiirille lasketaan määrävälein mittaus-, ohjaus- ja asetusarvosignaaliin sekä malliparametreihin perustuvia suorituskykyindeksejä, jotka esitetään säätötimanttina [Mar99]. Lisäksi tunnistetaan mittauksen ja ohjauksen saturoituminen ja muita hyödyllisiä säätöpiiriin liittyviä tietoja. LoopBrowserilla on seurattu hyvin tuloksin säätöpiirejä useissa teollisuusprosesseissa [Fri01].

LoopBrowser-ohjelmiston osaksi on kehitetty myös historiadataan perustuva

analyysityökalu LBhistory, jonka avulla voidaan tehdä automaattisesti määrävälein suorituskykyraportti useasta säätöpiiristä tietyltä aikajaksolta. LBhistorystä kerrotaan tarkemmin luvussa 6.

Muita työkaluja säätöpiirin suorituskyvyn seurantaan ovat ABB:n kehittämä AdvaControl LoopTuner, Fisher-Rosemountin kehittämä DeltaV Inspect, Honeywellin LoopScout-palvelu, Aspentech:in kehittämä Aspen Watch ja Matrikon:in kehittämä ProcessDoc [Air01].

2.4 Suorituskyvyn seuranta yksikköprosessitasolla

Yksikköprosessilla tarkoitetaan tuotantoprosessin rajattua toiminnallista kokonaisuutta, joka on siis itsekin prosessi. Yksikköprosessitaso on seuraava haaste automaattisten kunnonvalvonta- ja suorituskyvyn seurantajärjestelmien kehityksessä. Raja yksikköprosessin ja laitteen välillä ei kuitenkaan ole selvä, koska kaikki järjestelmät koostuvat osajärjestemistä. Jokin tärkeä prosessin osa saatetaan mieltää joko laitteeksi tai yksikköprosessiksi ilman mitään selvää kriteeriä. Tässä diplomityössä noudatetaan jaottelua, jossa yksikköprosessin täytyy koostua useasta tehtaan automaatiojärjestelmään kytketystä säätöpiiristä, jotka taas koostuvat laitteista.

Yksittäisten säätöpiirien analyysi ei kerro kovinkaan paljon prosessikokonaisuuden toiminnasta, koska yksittäinen nopeaksi viritetty säätöpiiri saattaa aiheuttaa häiriöitä muille säätöpiireille. Säätöpiirejä virittäessä täytyykin huomioida virityksen vaikutus prosessikokonaisuuteen.

Yksikköprosessitason suorituskykyä määritettäessä ongelmaksi muodostuu se, että eri yksikköprosessit ovat keskenään erilaisia, joten niiden suorituskykyä ei voida arvioida samoilla kriteereillä.

Yksikköprosessitason diagnostiikkaa ja suorituskykyä on tutkittu muunmuassa Oulun yliopistossa 'Osaprosessien Vikadiagnostiikka (OVI)'- projektissa ja VTT:llä 'Hierarkinen analysointityökalu tuotantolinjan osaprosessien tarkkailuun (HALOT)'-projektissa.

OVI-projektissa yksikköprosesseiksi miellettiin TMP-laitoksella sijaitseva massan lajittelu sekä rejektijauhatus. Lajitteluprosessissa latenssinpoistotornilta tuleva massa lajitellaan useita lajittimia käyttäen kiekkosuotimille syötettävään akseptiin ja uudelleen jauhettavaan rejektiin. Yksikköprosessin suorituskykyä mitataan prosessikohtaisilla järjestelmän tuottamilla massarejektisuhteella sekä sakeutumiskertoimella. Rejektijauhatuksessa lajitteluprosessin tuottama rejekti jauhetaan uudestaan ja syötetään takaisin lajitteluprosessiin. Tämän suorituskykyä mitataan laatuindeksillä ja stabiilisuusindeksillä, jotka ovat järjestelmän tuottamia tunnuslukuja [Ter02, Hil02].

HALOT-projektissa yksikköprosessijako tehtiin jakamalla paperikone toiminnallisiin lohkoihin. Projektissa kehitettiin hierarkinen menetelmä tuotantolinjan yksikköprosessien suorituskyvyn seurantaan. HALOT-järjestelmässä pyritään jäljittämään tuotantolinjan toiminnassa ilmenevä häiriö hierarkian avulla säätöpiirin viritykseen tai säätöpiirin osana

olevaan laitteeseen [Läh02].

HALOT-järjestelmässä tuotantolinjan suorituskyvyn oletetaan riippuvan prosessin hierarkisesti alempien toimintojen suorituskyvystä. Tässä lähestymistavassa yksikköprosessitason suorituskyky on siis jonkinlainen yhteenveto siihen kuuluvien toimintojen suorituskyvystä [Kiv02a, Kiv02b, Läh02].

Myös LoopBrowser ohjelmiston avulla voidaan seurata reaaliajassa yksikköprosessiin kuuluvien säätöpiirien suorituskykyä, joka osaltaan kertoo yksikköprosessi kokonaisuuden toiminnasta. LBhistory:n avulla voidaan laskea yksikköprosessikohtainen 'keskiarvo' siihen kuuluvien säätöpiirien suorituskyvystä tietyllä aikavälillä [Met02a, Met02b].

OY Keskuslaboratorio AB eli KCL on kehittänyt WEDGE-ohjelmiston, jolla pystytään analysoimaan prosessissa ilmeneviä värähtelyjä [Saa02]. Prosessien suorituskyvyn seurantaa tutkitaan myös tutkimuslaitoksessa nimeltä "Centre for Process Analytics and Control Technology (CPACT)", jossa on tehty joitakin MSPC-menetelmiin ja niiden laajennuksiin perustuvia prototyyppejä jäsenyritysten prosesseihin [Mar02].

Yksikköprosessin käyttäytymista voidaan verrata myös siitä laadittuun malliin. Esimerkiksi voimalaitosprosessi on mahdollista mallintaa suhteellisen tarkasti esimerkiksi APROS-ohjelmistolla [Yli01]. Yksikköprosesseja voidaan mallintaa myös neuroverkoilla. Metso Automationin kehittämän NeuroSense-ohjelman tuottama neuroverkkomalli voidaan ladata Metson automaatiojärjestelmään, jossa sitä voidaan hyödyntää [Pen01].

2.5 Kokemuksia kunnossapidon tukijärjestelmistä

Kunnossapidon tukijärjestelmiä on yritetty myydä tuotantolaitoksiin, mutta suurista lupauksista ja kiinnostuksesta huolimatta niiden kaupallistaminen on osoittautunut vaikeaksi. Syynä tähän on ollut ohjelmistojen suuri lukumäärä, järjestelmäriippuvuus, kallis hinta ja monimutkainen käyttöönotto.

Olemassaolevat järjestelmät ovat toimivia ja niiden käytöstä on saatu positiivisia kokemuksia [Fri01, Str01, Rau02, Lep02]. Kunnonvalvontaa onkin alettu myydä palveluna, jossa automaatiotoimittaja ja kunnossapitohenkilöstö huolehtivat kunnonvalvontajärjestelmien, etäyhteyksien ja palvelukeskusten avulla siitä, että tuotantolaitoksen prosessi toimii parhaalla mahdollisella tavalla.

Yksikköprosessitason monitorointijärjestelmien todellisesta toiminnasta on vaikea saada käsitystä, koska tuotantoprosessia ei yleensä haluta häiritä järjestelmän testauksen takia. Suorituskyvyn seurantajärjestelmiä kehittäessä on havaittu, että prosessien eri ajotilanteet ja erilaiset ajotavat häiritsevät suorituskyvyn määritystä, koska prosessilta vaaditaan erilaista käyttäytymistä eri tilanteissa. Prosessin ajotilanne pitäisikin ottaa huomioon suorituskykyä analysoitaessa.

Hierarkiassa ylöspäin mentäessä mittaustiedon pakkaaminen, jalostaminen ja

keskiarvoistaminen asettaa laskennassa käytetyille menetelmille kovan haasteen, koska laitetasolla tapahtuva yksittäisessä anturissa tai toimilaitteessa ilmenevä häiriö pitäisi kaikesta signaalinkäsittelystä huolimatta huomata ylemmän tason järjestelmissä. Lisäksi säätöpiirit 'piilottavat' ongelmia, jolloin niiden havaitseminen pelkästään mittaussignaaleja seuraamalla on mahdotonta [Läh02].

3 Klusterointimenetelmiä

3.1 Yleistä klusterianalyysistä

Datan klusterianalyysillä tarkoitetaan sekä suuridimensioisessa datassa mahdollisesti olevien ryhmien eli klustereiden etsimistä että yksittäisten näytteiden luokittelua löydettyihin klustereihin. Klusterointimenetelmät ovat myös kvantisointimenetelmiä, eli niillä voidaan mallintaa suuri määrä näytteitä pienemmällä määrällä koodivektoreita. Klusterianalyysi on tärkeä datamining-tekniikka, joka tuottaa informaatiota suuridimensioisen datan rakenteesta.

Klusterointimenetelmät ovat iteratiivisia algoritmeja, joilla pyritään sijoittamaan näyteavaruuteen koodivektoreita v, joiden avulla näytteet voidaan kuvata mahdollisimman tarkasti. Prosessia, jolla koodivektorit sijoitetaan, kutsutaan opetukseksi.

Kirjallisuudesta löytyy runsaasti erilaisia datan klusterointimenetelmiä [Bot95, Bez92, Dem77, Jai88, Koh95, Kau90, Mar93, McQ67]. Tässä luvussa esitellään viisi itseorganisoituvaa menetelmää. Niille kaikille on yhteistä kiinteä määrä koodivektoreita, jotka hakeutuvat datajoukossa piileviin ryhmiin yrittäen minimoida kvantisointivirheen E

$$E(\boldsymbol{V}) = \frac{1}{2}\sum_{i=1}^{n} d(x(i), v^c)^2 \tag{3.1}$$

jossa v^c on lähin koodivektori näytteelle $x(i)$ ja $\boldsymbol{V}$ on koodivektoreiden muodostama matriisi. Näytteen x ja koodivektorin v etäisyys d lasketaan yleensä käyttäen euklidista etäisyyttä (3.2) tai Mahalanobis-etäisyyttä (3.3).

$$d(x(i),v^c)^2=(x(i)-v^c)(x(i)-v^c)^T \tag{3.2}$$

$$d(x(i),v^c)^2=(x(i)-v^c)\boldsymbol{R}_c^{-1}(x(i)-v^c)^T \tag{3.3}$$

jossa $\boldsymbol{R}_c$ on näytteiden kovarianssimatriisi.

Yleisesti klusterointi on huonosti määritelty ongelma, koska näytteet muodostavat harvoin selviä klustereita. Ratkaisu, joka minimoi kvantisointivirheen E ei välttämättä ole klusteroinnin kannalta oikea [Fri00].

3.2 Datan esikäsittely

Analysoitavan datan esikäsittelyllä voidaan parantaa klusteroinnin tulosta. Esikäsittely on sovelluskohtainen tehtävä, jossa vaaditaan ymmärrystä tutkittavasta ilmiöstä. Yleensä esikäsittely koostuu muuttujien valinnasta, uusien piirteiden muodostuksesta, virheellisten näytteiden poistosta, kvalitatiivisten muuttujien koodauksesta ja eri mittayksiköissä olevien muuttujien normalisoinnista.

Esikäsittelyn vaativin vaihe on piirteiden muodostus. Piirteellä tarkoitetaan jotakin sovelluskohtaista, mahdollisesti useasta muuttujasta laskettavaa tunnuslukua, joka kuvaa tutkittavaa ilmiötä alkuperäisiä muuttujia paremmin.

Piirrevektoriin valitaan ainoastaan klusteroinnin kannalta oleelliset muuttujat tai näistä muodostetut piirteet. Ylimääräiset keskenään korreloivat ja epäoleelliset muuttujat kasvattavat piirrevektorin dimensiota ja häiritsevät klusterointia lisäämällä häiriöiden määrää.

Kvalitatiivisille muuttujille ei ole määritelty etäisyyttä eikä vertailuoperaatioita. Tällainen data täytyy muuntaa klusterointimenetelmille sopivaan muotoon esimerkiksi koodaamalla se tietyllä määrällä bittejä siten, että jokainen muuttujan arvo muodostaa oman binäärimuuttujan. Binäärimuuttuja saa arvon yksi, kun alkuperäisellä muuttujalla on binäärimuuttujaa vastaava arvo.

Analyysi perustuu etäisyyksien vertailuun, joten suuremmat lukuarvot omaava muuttuja dominoi analyysiä. Mittayksiköiden vaikutus voidaan poistaa näytejoukon normalisoinnilla,

$$x_j'(i) = \frac{x_j(i) - \mu_j}{\sigma_j} \quad \forall i = 1...n, j = 1...p \tag{3.4}$$

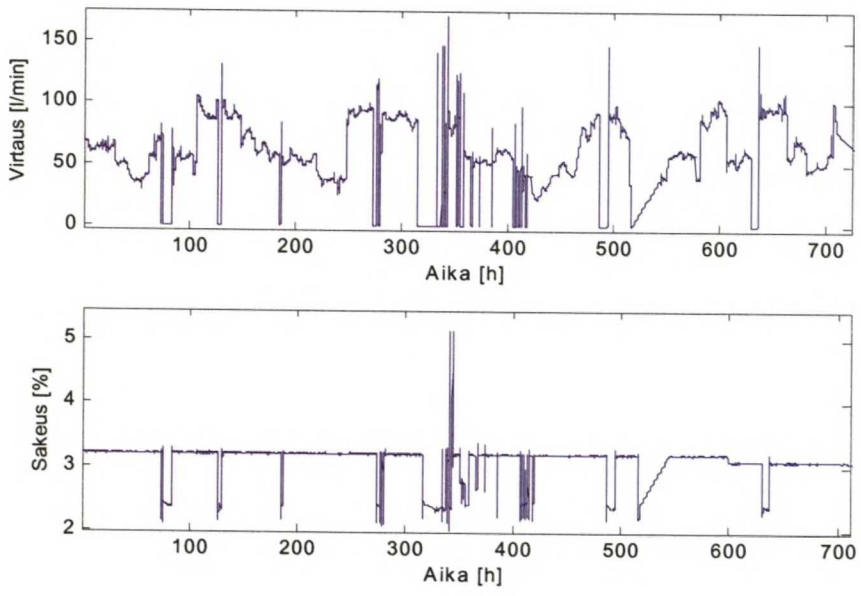
jossa $x_j(i)$ on alkuperäinen näyte muuttujasta j, $x'_j(i)$ on normalisoitu näyte, μ_j on muuttujan j aritmeettinen keskiarvo ja σ_j on muuttujan j keskihajonta [Mil95, Lai00].

Normalisoidun muuttujan keskiarvo on nolla ja varianssi yksi. Normalisoitu data on siis origon ympäristössä, joten tämän perusteella myös koodivektorit kannattaa aluksi sijoittaa origon ympäristöön. Normalisoinnin vaikutusta muuttujiin on havainnollistettu kuvissa 2 ja 3.

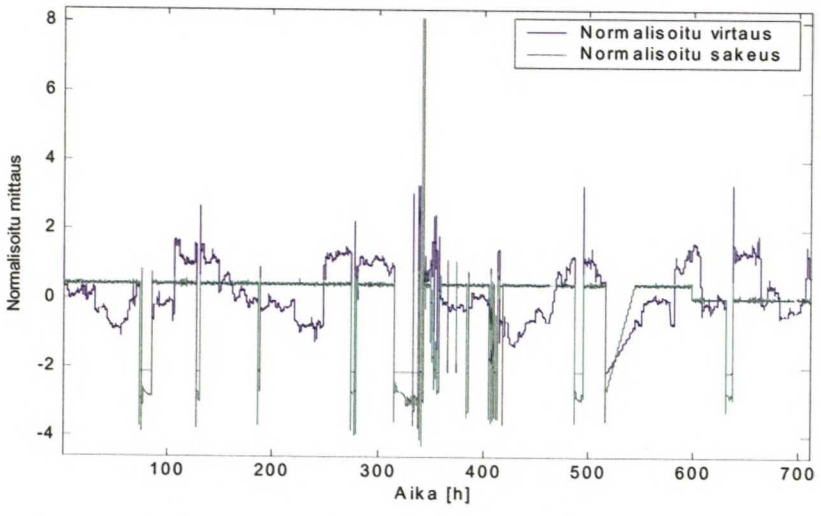
Signaalin vaihtelu koostuu hyödyllisestä informaatiosta ja häiriöistä. Eri muuttujilla kuvataan eri fysiikan suureita, joten niitä mitataan erilaisilla mittausperiaatteilla. Toisista suureista saadaan parempilaatuisia mittaustuloksia kuin toisista, jolloin on epärealistista olettaa, että kaikkien muuttujien informaatio olisi yhtä luotettavaa.

Varianssi on suurempi muuttujalle, jossa on paljon suuria tasomuutoksia ja vähän kohinaa kuin muuttujalle, jossa on paljon kohinaa ja pieniä tasomuutoksia. Normalisointi aiheuttaa sen että pienitasoisessa muuttujassa oleva kohina vahvistuu suuritasoisen muuttujan kustannuksella.

Eri muuttujien erilaiset signaali/kohinasuhteet aiheuttavat sen, että pelkkä normalisointi ei välttämättä riitä vaan tarvitaan painokertoimia, joilla luotettavampia muuttujia voidaan painottaa enemmän. Näiden valintaan tarvitaan kuitenkin tietoa eri muuttujien luotettavuudesta.



Kuva 2 Paperikoneen massaosaston eräästä linjasta kerättyä mittausdataa. Ylemmässä kuvassa on esitetty virtaus ja alemmassa kuvassa massan sakeus. Virtaus saa luku arvoja 0-150 l/min ja sakeus 0-5%.



Kuva 3 Samat signaalit esitetty normalisoituna. Molempien signaalien lukuarvot ovat –4:n ja 4:n välillä. Sakeuden merkitys klusterianalyysin tulokseen kasvaa merkittävästi. Huomaa eriyisesti hetkellä *t*=600 ilmenevä sakeuden asetusarvomuutos. Normalisoinnin ansiosta sakeusmittauksessa esiintyvän kohinan merkitys kasvaa suhteessa virtausmittauksessa oleviin vaihteluihin.

3.3 Klusterointialgoritmeja

3.3.1 K-means

Olkoon näytejoukko n x p matriisi, eli se sisältää n kappaletta p-dimensioisia näytevektoreita x. Algoritmissa näytejoukon oletetaan sisältävän k kappaletta samankaltaisten näytteiden ryhmiä eli klustereita Γ. Jokaiselle klusterille Γ määrätään koodivektori v, joka on näyteryhmää kuvaava esimerkkinäyte. Koodivektorit muodostavat koodikirjan V, joka on k x p-matriisi. Ideana on sijoittaa eri ryhmien koodivektorit näyteavaruuteen siten, että kvantisointivirhe (3.1) minimoituu.

K-means algoritmin [McQ67, Hyö01] suoritus etenee seuraavasti:

1. Valitaan klustereiden lukumäärä ja annetaan koodivektoreille v alkuarvot.

2. Määrätään kaikki n näytettä klustereihin Γ käyttäen kriteerinä euklidista etäisyyttä:

$$x(i) \in \Gamma^c \text{ jos } \left\| x(i) - v^c \right\| = \min_j \left\{ \left\| x(i) - v^j \right\| \right\} \quad \forall\, j = 1...k \quad \forall i = 1...n \tag{3.5}$$

3. Siirretään kaikkien klustereiden koodivektorit klusteriin kuuluvien näytteiden aritmeettisen keskiarvon määräämään pisteeseen:

$$v^j = \frac{\sum_{x \in \Gamma^j} x}{\#\{\Gamma^j\}} \quad \forall j = 1...k \tag{3.6}$$

jossa $\#\{\Gamma^j\}$ tarkoittaa näytteiden lukumäärää klusterissa Γ^j.

4. Jos jonkun klusterin koodivektori muuttui, palataan kohtaan 2. Muussa tapauksessa algoritmin suoritus lopetetaan.

K-means algoritmista tunnetaan myös rekursiivinen versio, jossa iteraatiolla käytetään vain yhtä näytettä $x(t)$. Tämä näyte määrätään johonkin klusteriin, jonka koodivektoria päivitetään rekursiivisen keskiarvon menetelmällä. Rekursiivisessa algoritmissa klusterille on määritelty koodivektorin lisäksi paino h, joka kasvaa aina kun näyte osuu klusteriin.

Rekursiivinen versio etenee seuraavasti:

1. Valitaan klustereiden lukumäärä ja annetaan koodivektoreille v alkuarvot
2. Määrätään näyte $x(t)$ klusteriin Γ^c käyttäen kriteerinä euklidista etäisyyttä:

$$x(t) \in \Gamma^c \text{ jos } \left\| x(t) - v^c \right\| = \min_j \left\{ \left\| x(t) - v^j \right\| \right\} \ \forall\ j = 1...k \tag{3.7}$$

3. Siirretään lähimmän klusterin Γ^c koodivektoria v^c kohti näytettä:

$$v^c(t+1) = \frac{h_c v^c(t) + x(t)}{h_c + 1} \tag{3.8}$$

jossa h_c on näytteiden lukumäärä klusterissa Γ^c.

4. $h_c = h_c + 1$ (3.9)
5. $t = t + 1$, Palataan kohtaan 2.

3.3.2 Itseorganisoituva kartta (SOM)

Itseorganisoituvaa karttaa (Self-organizing Map) käytetään suurien ja monidimensioisten datajoukkojen tulkintaan ja visualisointiin. Menetelmä mielletään neuroverkoksi, joten koodikirjan elementtiä kutsutaan neuroniksi. Neuronille on määritelty koodivektorin lisäksi sen sijainti kaksiulotteisella kartalla [Koh95].

Itseorganisoituva kartta on siis kaksiulotteinen neuroneiden taso, johon suuridimensioinen data kuvataan siten, että datassa olevat rakenteet säilyvät. Kuvaus tehdään sijoittamalla neuroneiden koodivektorit näyteavaruuteen siten, että kvantisointivirhe E minimoituu.

SOM-algoritmi etenee seuraavasti:

1. Päätetään kartan koko ja annetaan neuroneiden koodivektoreille alkuarvot.
2. Otetaan näytejoukosta satunnaisesti näytevektori $x(t)$.
3. Etsitään se neuroni Γ^c, jonka koodivektori v^c on lähimpänä näytettä $x(t)$:

$$x(t) \in \Gamma^c \text{ jos} \left\| x(t) - v^c \right\| = \min_j \left\{ \left\| x(t) - v^j \right\| \right\} \ \forall\ j = 1...k \tag{3.10}$$

4. Kaikkien neuroneiden koodivektoreita v päivitetään:

$$v^j(t+1) = v^j(t) + h_{cj}(t)\left[x(t) - v^j(t) \right] \ \forall\ j = 1...k \tag{3.11}$$

$$h_{cj}(t) = \alpha(t)\exp\left(-\frac{\|r_c - r_j\|}{2\sigma(t)^2}\right) \tag{3.12}$$

jossa $\alpha(t)$ on oppimiskerroin, joka vähenee ajan kuluessa ja $\sigma(t)$ määrää gaussisen naapuruusfunktion leveyden.

5. Palataan kohtaan 2.

Yhtälössä 3.12 esiintyvä $\|r_c - r_j\|$ tarkoittaa näytevektoria lähimmän neuronin I^c etäisyyttä neuroniin I^j kaksiulotteisella kartalla naapuruussuhteet huomioiden.

Koodikirjan päivityksessä yhden näytteen perusteella päivitetään useita koodivektoreita. Neuronit on sidottu toisiinsa naapuruussuhteella, joten näyteavaruudessa toisiaan lähellä olevat neuronit ovat lähekkäin myös kartalla.

3.3.3 Neural gas

Neural gas algoritmissa päivitetään yhden näytteen perusteella useampia koodivektoreita. Perusidea on, että päivityksen suuruus ei riipu suoraan etäisyydestä kuten SOM-algoritmissa, vaan neuronin järjestysnumerosta. Näytteen etäisyys jokaisen neuronin koodivektoriin lasketaan, jonka jälkeen neuronit järjestetään etäisyyden mukaan. Tätä operaatiota kutsutaan nimellä "Neighbourhood ranking" [Mar93].

Algoritmi etenee seuraavasti:

1. Valitaan neuroneiden lukumäärä ja annetaan niiden koodivektoreille alkuarvot.

2. Otetaan näytejoukosta näytevektori $x(t)$.

3. Lasketaan kaikille neuroneille etäisyys näytevektorin ja neuronin koodivektorin välille ja järjestetään neuronit etäisyyden perusteella kasvavaan järjestykseen ("Neighbourhood ranking")

4. Päivitetään kaikkien neuroneiden koodivektorit:

$$v^j(t+1) = v^j(t) + \varepsilon \cdot h_\lambda\left(k_j(x(t), V(t))\right) \cdot \left(x(t) - v^j(t)\right) \forall\ j = 1...k \tag{3.13}$$

5. $t = t+1$, Palataan kohtaan 2.

Kohdassa neljä tarvittava ε on päivityksen voimakkuuteen liittyvä vakio. Järjestysluku lasketaan funktiolla $k_j(x(t), V(t))$, jossa k_j on neuronin j saama järjestysluku siten, että koodikirjassa V on k-1 neuronia, joiden koodivektori ovat lähempänä näytevektoria $x(t)$.

Järjestysluvun vaikutus päivityksen voimakkuuteen määrätään funktiolla h_λ.

Päivitys ei riipu suoraan etäisyydestä kuten muissa algoritmeissa vaan järjestysluvusta k_j. Neuroneiden välille ei ole määritetty naapuruussuhteita kuten SOM-algoritmissa, vaan kaikki neuronit ovat toistensa naapureita. Naapuruussuhde riippuu järjestysluvusta joten yksittäinen näyte vaikuttaa kaikkiin neuroneihin 'kaasumaisesti'.

3.3.4 Expectation Maximization (EM)

Expectation Maximization algoritmissa näytejoukon oletetetaan koostuvan eri tavalla normaalijakautuneista (3.14) näytejoukoista (Gaussian mixtures). Klusterilla tarkoitetaan siis tiettyä normaalijakaumaa noudattavaa näytejoukkoa

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^p \det\{\boldsymbol{R}\}}} e^{-\frac{1}{2}(x-\bar{x})^T R^{-1}(x-\bar{x})} \tag{3.14}$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x(i) \tag{3.15}$$

$$\boldsymbol{R} = \frac{1}{n}\sum_{i=1}^{n}(x(i)-\bar{x})(x(i)-\bar{x})^T \tag{3.16}$$

Yksittäiselle näytteelle voidaan laskea todennäköisyys, jolla se kuuluu tiettyyn klusteriin. Jokaiselle näytteelle haetaan se klusteri, jonka todennäköisyysjakauma tuottaa näytteelle suurimman todennäköisyyden (Maximum likelihood) [Dem77,Hyö01].

Laskennan helpottamiseksi todennäköisyysjakauman tiheysfunktiosta kannattaa ottaa luonnollinen logaritmi, jolloin saadaan kriteeri näytteen $\boldsymbol{x}(i)$ kuulumisesta klusteriin j (3.17).

$$\log(p(x(i))) = -\frac{p}{2}\cdot\log(2\pi) - \frac{1}{2}\cdot\log\left(\det\{\boldsymbol{R}^j\}\right) - \frac{1}{2}\left(x(i)-v^j\right)^T\left(\boldsymbol{R}^j\right)^{-1}\left(x(i)-v^j\right) \tag{3.17}$$

jossa ensimmäinen termi on kaikille klustereille yhtäsuuri vakio ja viimeinen termi on Mahalanobis-etäisyys (3.3).

Suuri varianssi tietyssä signaaliavaruuden suunnassa vähentää etäisyyden merkitystä tässä suunnassa suhteessa sellaisiin suuntiin, joissa varianssi on pienempi. Tämän ominaisuuden avulla datajoukosta on mahdollista löytää elliptisiä dataryhmiä (kuva 11). EM-menetelmässä klusterilla Γ takoitetaan siis olioa, jolle on määritelty koodivektori v ja kovarianssimatriisi $\boldsymbol{R}$.

Algoritmi etenee seuraavasti:

1. Valitaan klustereiden lukumäärä ja annetaan niiden koodivektoreille alkuarvot. Klustereiden kovarianssimatriisit $\boldsymbol{R}$ asetetaan yksikkömatriikseiksi.

2. Määrätään kaikki näyteet $x(i)$, i=1...n klustereihin käyttäen kriteerinä loglikelihood-funktiota (3.17).

$$x(i) \in \Gamma^c, \text{ jos } \log\left(\det\left\{\boldsymbol{R}^c\right\}\right) + d\left(x(i), v^c\right) = \min_j \left\{\log\left(\det\left\{\boldsymbol{R}^j\right\}\right) + d\left(x(i), v^j\right)\right\} \forall j = 1...k \tag{3.18}$$

3. Päivitetään klustereiden koodivektorit ja kovarianssimatriisit:

$$v^j \leftarrow \frac{\sum_{x(i)\in\Gamma^j} x(i)}{\#\left\{\Gamma^j\right\}} \forall\ j = 1...k \tag{3.19}$$

$$\boldsymbol{R}^j \leftarrow \frac{\sum_{x(i)\in\Gamma^j} \left(x(i) - v^j\right)\left(x(i) - v^j\right)^T}{\#\left\{\Gamma^j\right\}} \forall\ j = 1...k \tag{3.20}$$

4. Jos jonkun klusterin koodivektori muuttuu, palataan kohtaan 2. Muussa tapauksessa algoritmin suoritus lopetetaan.

Jos kovarianssimatriisia pidetään koko ajan yksikkömatriisina, algoritmi on täysin sama kuin K-means.

3.3.5 Fuzzy c-means

Sumea klusterointi eroaa muista tässä esitetyistä algoritmeista siten, että näyte kuuluu kaikkiin klustereihin jollakin jäsenyysasteella, joka riippuu näytteen ja koodivektorin etäisyydestä.

Algoritmi etenee seuraavasti [Bez92]:

1. Valitaan klustereiden lukumäärä ja annetaan niiden koodivektoreille alkuarvot.

2. Lasketaan kaikille näytteille $x(i)$ etäisyys kaikkiin klustereihin käyttäen Euklidista etäisyyttä.

$$d_{i,j}^{\ 2} = \left(x(i) - v^j\right)\left(x(i) - v^j\right)^T \ \forall\ i = 1...n, j = 1...k \tag{3.21}$$

3. Määritetään kaikille näytteille $x(i)$ jäsenyysasteet $u_{i,j}$ kaikkiin klustereihin

$$u_{i,j} = \frac{1}{\sum_{s=1}^{k} \left(\frac{d_{i,j}}{d_{i,s}} \right)^{\frac{2}{m-1}}}, \forall\, i = 1...n, j = 1...k \tag{3.22}$$

$$\sum_{j=1}^{k} u_{i,j} = 1 \;\forall\, i = 1...n \tag{3.23}$$

4. Päivitetään klustereiden koodivektorit

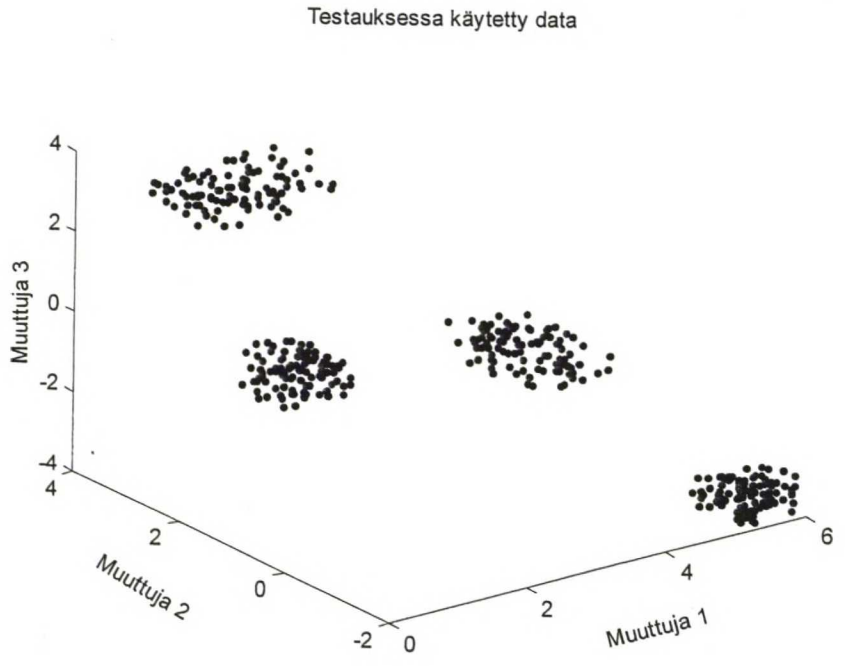
$$v^j = \frac{\sum_{i=1}^{n} x(i)\left(u_{i,j}\right)^m}{\sum_{i=1}^{n} \left(u_{i,j}\right)^m} \;\forall\, j = 1...k \tag{3.24}$$

5. Palataan kohtaan 2.

Yhtälöissä esiintyvä m on käyttäjän määräämä sumeutukseen liittyvä parametri, jonka täytyy olla suurempi kuin yksi. Jokaiselle näytteelle täytyy siis laskea erikseen etäisyys ja jäsenyysaste jokaiseen klusteriin. Online-luokittelussa näytteiden jäsenyysasteet eri klustereihin voidaan laskea vaiheen kolme kaavalla (3.22).

3.4 Algoritmien ominaisuuksia

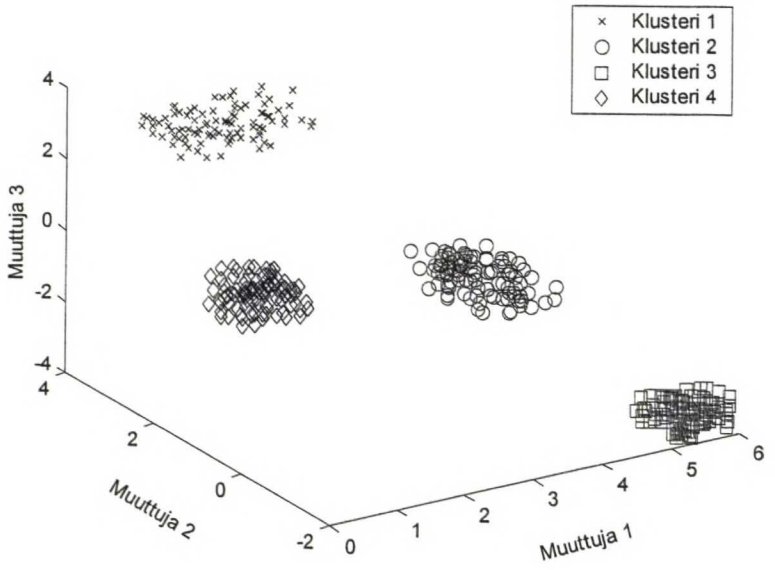
Kaikki viisi algoritmia on toteutettu Matlab-ympäristössä ja niitä testattiin kolmesta signaalista muodostetulla testidatalla, joka on esitetty kuvassa 4. Data on generoitu satunnaislukugeneraattorilla, joten eri signaaleilla ei ole mittayksiköitä. Datassa on 400 näytettä, jotka jakautuvat neljään klusteriin. Jokaisessa klusterissa on 100 näytettä.



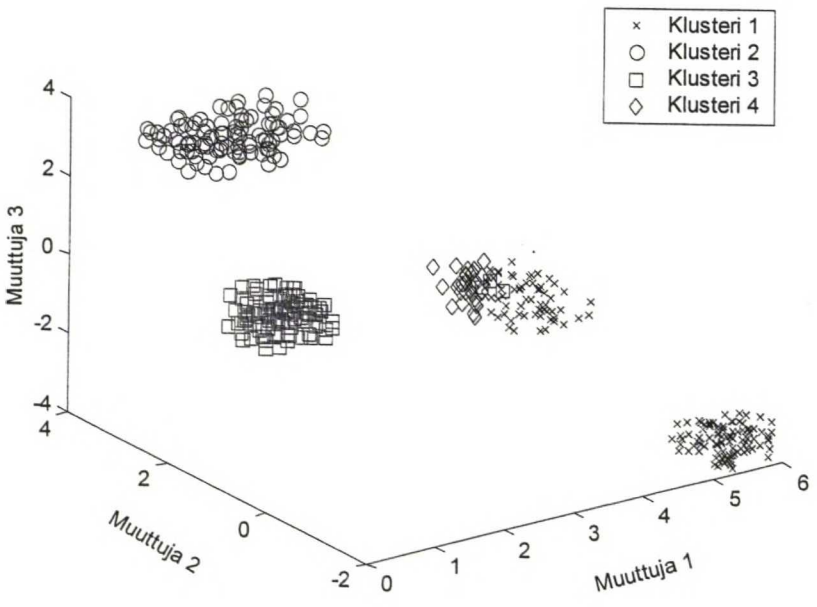
Kuva 4 Algoritmien kokeilussa käytetty kolmiulotteinen testidata. Datassa on neljä selvää klusteria, jotka yritetään löytää eri algoritmeilla. Data on luotu satunnaislukugeneraattorilla, joten signaaleilla ei ole mittayksiköitä.

K-means algoritmi minimoi kvantisointivirheen $E(V)$ käyttäen Newtonin gradienttimenetelmää [Bot95]. Gradienttimenetelmä päätyy lokaaliin minimikohtaan, joka vaihtelee eri alkuarvoilla. Algoritmia kokeiltiin testidatalle ja tulokset on esitetty kuvissa 5 ja 6. K-means algoritmi on yksinkertainen toteuttaa, mutta sen antama tulos riippuu koodivektoreille annettavista alkuarvoista. K-means algoritmilla saadaan selvästi klusteroituneelle datalle eri tulos riippuen alkuarvoista.

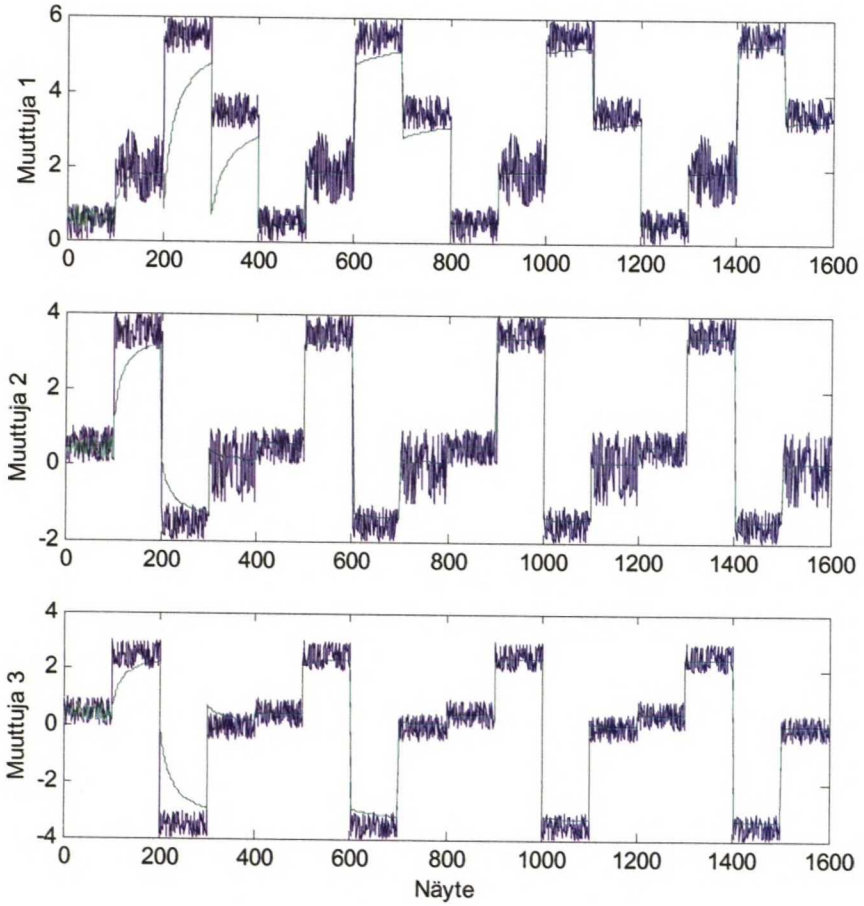
Rekursiivisessa K-means algoritmissa käsitellään kerrallaan yhtä näytettä, jonka perusteella päivitetään ainoastaan lähintä koodivektoria. Rekursiivista K-means algoritmia testattiin siten, että samaa dataa iteroitiin neljä kertaa. Koodivektorin sijainti jokaisessa dimensiossa, jokaisen näytteen jälkeen on esitetty kuvassa 7. Tulos kuitenkin riippuu koodivektoreiden alkuarvoista.



Kuva 5 Hyvillä alkuarvolla K-means algoritmi löytää ryhmät oikein.



Kuva 6 K-means algoritmin tulos saattaa olla huonoilla alkuarvoilla virheellinen. Klusterin 1 koodivektori on sijoittunut kahden ryhmän väliin. Myös klustereiden nimet muuttuivat.

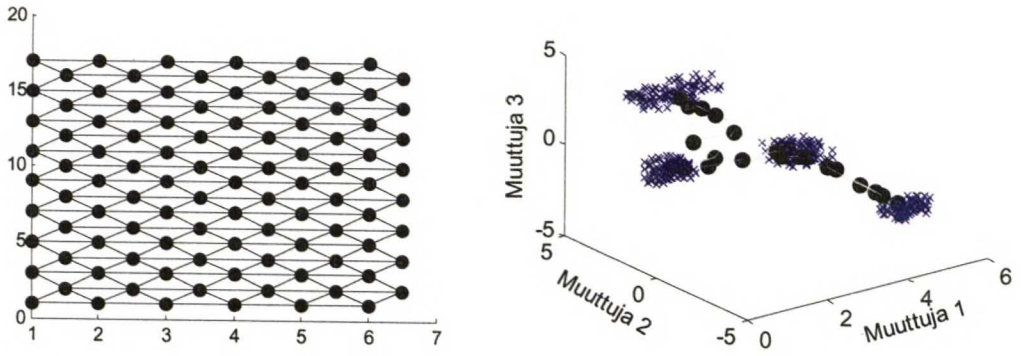


Kuva 7 Koodikirjan kehittyminen käyttäen rekursiivista K-means algoritmia. Näytteet vetävät lähintä koodivektoria itseään kohti, jolloin koodivektorit löytävät samankaltaisten näytteiden ryhmät.

Neural gas algoritmi on toteutettu Matlab ympäristöön SOM Toolbox-funktiokirjastossa [Alh00]. Algoritmia kokeiltiin testidatalle, jonka neljä klusteria löytyi ensimmäisellä iteraatiolla. Useista eri alkuarvoista huolimatta saatu tulos oli aina oikea.

Algoritmi on kuitenkin K-means algoritmiin verrattuna raskas suorittaa, koska jokaista näytettä kohti täytyy tehdä neighbourhood ranking-operaatio, jonka jälkeen päivitetään jokaista neuronia.

SOM-algoritmi eroaa muista klusterointimenetelmistä siten, että koodikirja voidaan visualisoida ymmärrettävästi ja koodikirjassa lähellä toisiaan olevat neuronit edustavat samankaltaisia näytteitä.

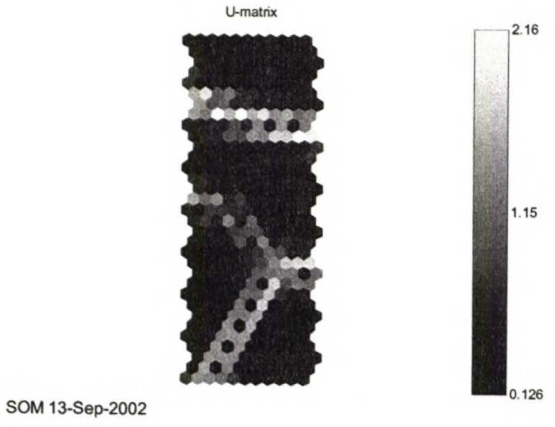


Kuva 8 Itseorganosoituva kartta muodostaa moniulotteisesta datasta kaksiulotteisen kartan, jossa samankaltaiset näytteet ovat lähellä toisiaan.

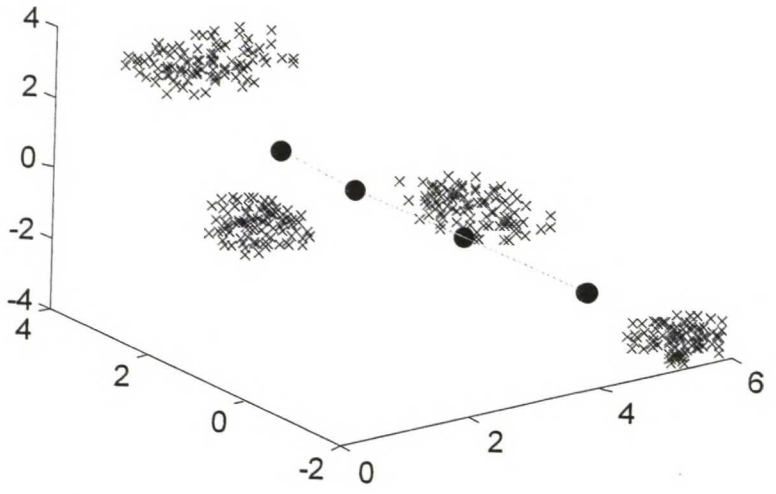
Kartan perusteella voidaan muodostaa erilaisiin tunnuslukuihin perustuvia kuvaajia. Eräs kuvaaja on nimeltää U-matriisi (Unified Distance Matrix), jolla kuvataan neuronien koodivektorien etäisyyttä toisistaan väreillä. Tällaisen kuvaajan avulla nähdään moniulotteisessa datassa mahdollisesti piilevät klusterit. Testidatalle piirretty U-matriisi on esitetty kuvassa 9.

Algoritmiin määritellyt naapuruussuhteet estävät koodivektoreiden vapaan liikkumisen, jolloin neuroneiden määrä pitää olla huomattavasti suurempi kuin datassa olevien ryhmien määrä. Tätä on havainnollistettu kuvassa 10. Tämä on klusterianalyysin kannalta huono ominaisuus, koska klusterianalyysissä löydettyjen ryhmien pitäisi kuvata jotain ilmiötä ja ryhmien määrä pitäisi olla mahdollisimman pieni.

Datassa olevien ryhmien lukumäärää voidaan kuitenkin arvioida U-matriisin avulla, josta saadaan hyvä arvio tarvittavien klustereiden lukumäärästä. Tämän jälkeen voidaan käyttää jotain muuta klusterointimenetelmää datan jakamiseksi luokkiin.

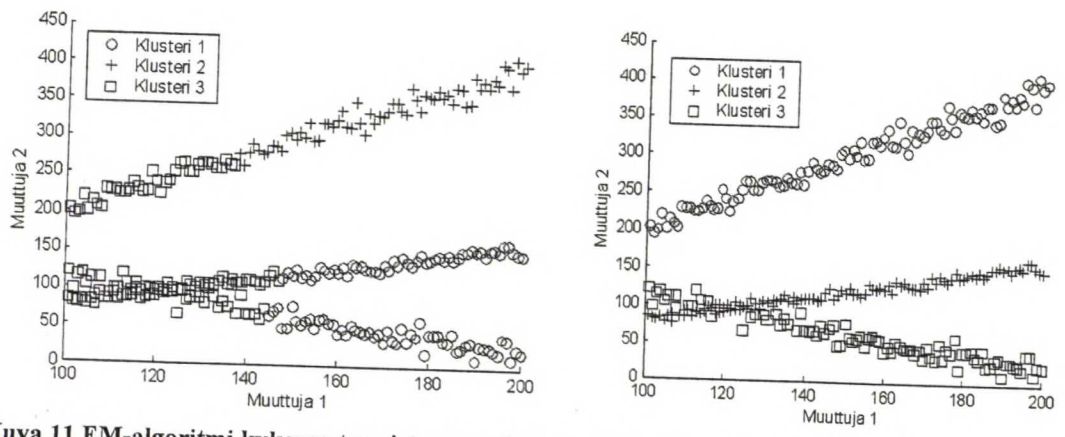


Kuva 9 Datan kluterirakennetta kuvaava U-matriisi. Tumma väri kuvaa lyhyttä etäisyyttä neuronien välillä. Kuvassa nähdään neljä erillistä tummaa aluetta, mikä tarkoittaa että alkuperäinen data on jakaantunut neljään klusteriin.



Kuva 10 SOM-algoritmin tulos kun testidataan neuroneiden lukumäärä oli neljä ja kartan koko 4x1. SOM-algoritmia käytettäessä neuroneiden lukumäärä pitää olla suuempi kuin datassa olevien ryhmien määrä, koska naapuruussuhteet estävät koodivektoreiden vapaan liikkumisen.

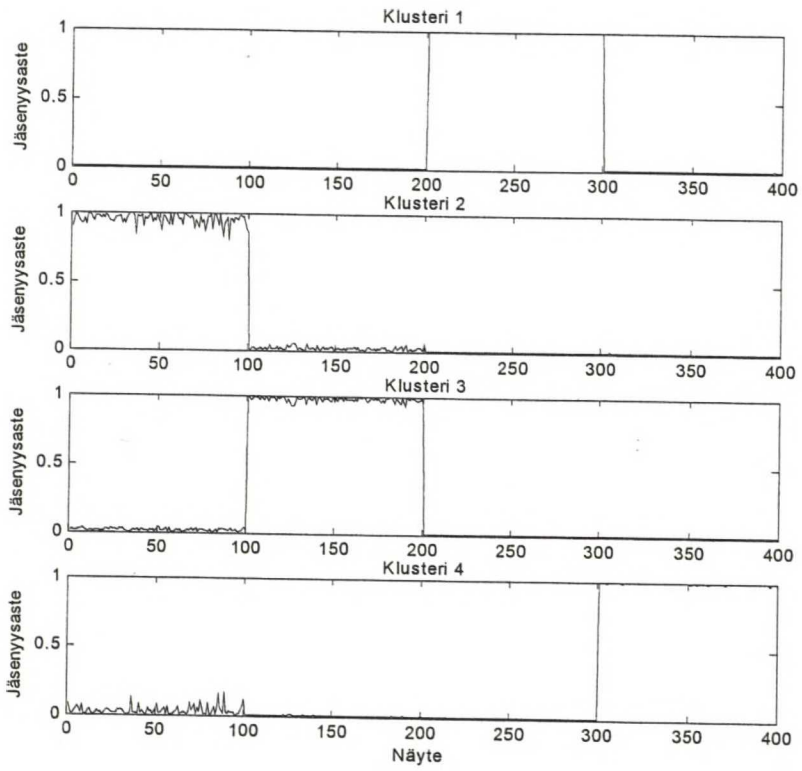
Expectation Maximization algoritmi kykenee löytämään datasta ellipsoidin muotoisia datajoukkoja, koska eri suunnissa olevaa etäisyyttä painotetaan eri tavalla. Kuvassa 11 on esitetty K-means ja EM-algoritmeilla saatu tulos eräälle datajoukolle. Algoritmi on erittäin herkkä alkuarvoille, pieni muutos kovarianssimatriisissa $\boldsymbol{R}$ voi muuttaa tuloksen aivan toisenlaiseksi.



Kuva 11 EM-algoritmi kykenee tunnistamaan datasta ellipsoidin muotoisia joukkoja. Vasemmalla on K-means –algoritmin antama euklidiseen etäisyyteen perustuva tulos ja oikella EM-algoritmin Maximum likelihood-kriteeriin perustuva tulos.

Fuzzy C-means algoritmi laskee näytteelle jäsenyysasteet kaikkiin klustereihin. C-means algoritmia kokeiltiin testidatalle parametriarvolla m=1.2 algoritmi löysi testidatan neljä klusteria kolmella iteraatiolla. Klustereiden jäsenyysasteet on esitetty kuvassa 12.

Lopuksi taulukkoon 1 on kerätty yhteenveto esiteltyjen klusterointimenetelmien ominaisuuksista.



Kuva 12 Näytteiden jäsenyysasteet eri klustereihin. Fuzzy C-means algoritmissa näyte kuuluu kaikkiin klustereihin, jollakin jäsenyysasteella. Tässä kokeessa näytteet 0-100 kuuluvat eniten klusteriin 2, näytteet 100-200 klusteriin 3, näytteet 200-300 klusteriin 1 ja 300-400 klusteriin 4.

Taulukko 1 Yhteenveto eri klusterointimenetelmien ominaisuuksista.

Algoritmi	Algoritmin tyyppi	Alkuarvo-riippuvuus	Tarvittavat parametrit	Algoritmin kompleksisuus	Erityisominaisuuksia
K-means	Batch/ Rekursiivinen	Suuri	Klustereiden lukumäärä	$O(kpn)$	-
SOM	Rekursiivinen	Pieni, jos neuroneita on paljon	Neuroneiden lukumäärä Kartan koko	$O(k^2pn)$	Koodikirja muodostaa kartan, jonne samankaltaiset näytteet kuvautuvat lähekkäin.
Neural gas	Rekursiivinen	Pieni	Neuroneiden lukumäärä Iteraatioiden lukumäärä	$O(k \log(k)pn)$	-
EM	Batch	Suuri	Klustereiden lukumäärä	$O(kp^2n)$	Näytteet luokitellaan klustereihin käyttäen Maximum likelihood-kriteeriä.
FCM	Batch	Pieni	Klustereiden lukumäärä Iteraatioiden lukumäärä Sumeutus m	$O(k^2pn)$	Näyte kuuluu kaikkiin klustereihin, jollakin jäsenyysasteella.

4 Ajotilanteen tunnistusjärjestelmä

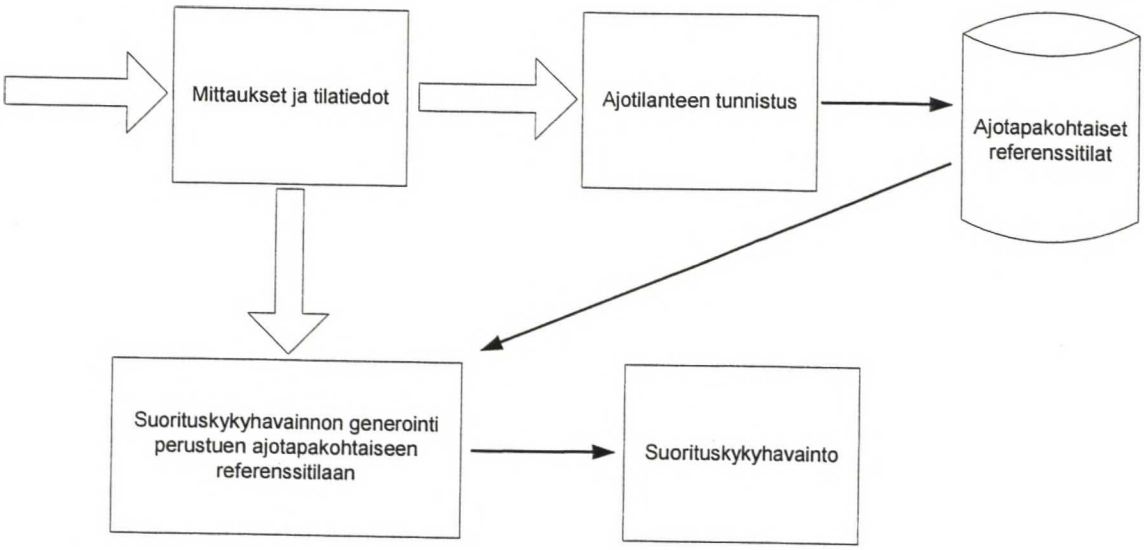
4.1 Yleistä ajotilanteen tunnistamisesta

Prosessin ajotilanteella tarkoitetaan erilaisia prosessissa vallitsevia olosuhteita, joiden vaihtelu aiheutuu esimerkiksi:

- Eri tuotantomääristä ja tuotantonopeuksista
- Eri lajien ja tuotteiden ajamisesta
- Raaka-aineen ominaisuuksien vaihtelusta
- Vika- ja häiriötilanteista
- Vuodenaikojen vaihtelusta
- Ylös- ja alasajotilanteista
- Panosprosessin eri vaiheista
- Erilaisista ajotavoista

Ongelmat kunnonvalvonnan ja suorituskyvyn seurantajärjestelmien toiminnassa johtuvat suurelta osin prosessin eri ajotilanteista, joita ei oteta laskennassa huomioon. Ajotilanne vaikuttaa yksikköprosessin suorituskyvyn seurantaan, koska yksikköprosessin hyvä suorituskyky tarkoittaa eri asioita eri ajotilanteissa.

Järjestelmillä halutaan myös tutkia onko esimerkiksi tuotantonopeudella tai erilaisilla tuotelajeilla vaikutusta tietyn yksikköprosessin suorituskykyyn. Yleiskäyttöisen järjestelmän täytyy siis pystyä tunnistamaan vallitseva ajotilanne prosessista saatavan mittaus- ja tilatietoinformaation perusteella. Ajotilanteille voidaan määrätä ajotilannekohtaiset referenssitilat, joita käytetään suorituskyvyn määrityksessä.



Kuva 13 Ajotilanteen tunnistus ja ajotilannetiedon hyödyntäminen suorituskyvyn seurannassa [Kiv01].

Ajotilanteen tunnistamiseksi on ehdotettu useita ratkaisuvaihtoehtoja [Kiv01]. Taulukointi, jossa mittauksille ja tilatiedoille annetaan selvät ajotilannekohtaiset raja-arvot, on teknisesti helpoin tapa. Rajat pitää kuitenkin tuntea ja todellisuudessa raja-arvot eivät ole selkeitä. Tunnistusta varten on toteutettu sumeaa logiikkaa hyödyntävä luokittelija, joka on osoittautunut toimivaksi. Ratkaisu vaatii kuitenkin jäsenfunktioiden parametroinnin, mikä on liian vaikea ja aikaavievä toimenpide yleisessä tapauksessa.

Myös tapauspohjaista päättelyä (Case Based Reasoning) on esitetty ratkaisuksi tunnistusongelmaan. Päättelykone koostuu rajallisesta määrästä aiempia tapauksia tai esimerkkejä, joihin kullakin hetkellä voimassa olevaa tilannetta verrataan. Mikäli mikään esimerkki ei sovi tilanteeseen, se lisätään esimerkkien joukkoon. Päättelykone ikäänkuin oppii esimerkkien kautta tunnistamaan eri tilanteet. CBR-menetelmässä vaaditaan raja-arvo, joka määrää onko tilanne uusi vai ei. Uudet tilanteet kuitenkin aiheuttavat ongelmia ylemmän tason sovelluksille, koska niillä ei ole tietoa kuinka uusia tilanteita pitäisi tulkita.

Prosessista saatavan datan klusterianalyysi on luonnollinen tapa löytää ja tunnistaa prosessin eri ajotilanteet. Prosessin tietylle ajotilanteelle on tyypillistä, että signaaliarvot esiintyvät tietyssä signaaliavaruuden osassa, jolloin ajotilanteet on mahdollista erottaa toisistaan klusterianalyysin avulla.

Myös kirjallisuudesta löytyy esimerkkejä klusterianalyysin käytöstä eri ajotilanteiden löytämiseksi. Klusterianalyysin avulla on tutkittu esimerkiksi vuodenajoista johtuvaa vaihtelua jäteveden puhdistusprosessissa [Tep99]. Klusterointiin perustuvaa luokittelua on onnistuneesti käytetty myös malmityypin online-tunnistuksessa rikastusprosessissa [Lai94].

4.2 Toteutetun ohjelmiston rakenne

Diplomityössä toteutettiin DNAhistorian-tietokantaympäristöön tietokantaan syötettävän datan klusterointiin perustuva ohjelmistopaketti. Se on toteutettu prosessin ajotilanteen tunnistamista varten, mutta sitä voidaan käyttää mihin tahansa datan klusterointia, kvantisointia tai luokittelua vaativaan sovellukseen. Ohjelmistopaketti koostuu Matlab-ympäristöön toteutetusta konfiguraattorista ja DNAhistorian tietokantaympäristöön toteutetusta laskentalohkosta. Lisäksi toteutettiin kvantisointivirheeseen perustuva Java-pohjainen diagnostiikkasovelluksen prototyyppi. Toteutetun ohjelmistopaketin rakennekaavio on esitetty liitteessä 1.

Järjestelmän online osa suorittaa rekursiivista klusterointialgoritmia tietokantaan syötettävälle datalle. Se on kirjoitettu DNAhistorian-tietokannan omalla ohjelmointikielellä, joka on laajennus SQL-standardiin [Sql92]. Laskentalohko koostuu proseduureista (procedure), konfiguraatiotietokannasta ja tulostietueista (record).

Konfiguraatiotietokantaan on tallennettu laskennassa tarvittavia tietoja ja parametreja kuten seurattavien signaalien tietokantaosoitteet, joiden perusteella ajotilanteen tunnistus tapahtuu. Tulostietueisiin kirjoitetaan laskennan tuloksena syntyvä ajotilannesignaali ja tunnistuksen luotettavuutta kuvaava etäisyys näytevektorin ja ajotilannetta kuvaavan klusterin koodivektorin välillä eli kvantisointivirhe.

Ohjelmalla on pääfunktio, jota kutsutaan osana erillistä DNAhistorian tehtäväsäiettä (Task) määrävälein esimerkiksi kerran minuutissa. Myös aktivointi jonkin tapahtuman seurauksena on mahdollista.

Offline-konfiguraattorin tarkoitus on helpottaa järjestelmän käyttöönottoa. Se on toteutettu Matlab-funktioina, joilla muokataan ODBC-yhteyksiä käyttäen konfiguraatiotietokantaa.

Diagnostiikkamoduuli seuraa klusterointimoduulin tuottamaa kvantisointivirhettä ja ilmoittaa mikäli kvantisointivirhe ylittää raja-arvon.

4.3 Klusterointilohkon toimintaperiaate

Automaatiojärjestelmästä kerättävistä näytteistä muodostettujen näytevektoreiden luokittelu perustuu koodikirjaan, joka sisältää esimerkkejä erilaisista ajotilanteista. Koodikirja voidaan antaa prosessituntemuksen perusteella, jollakin offline-klusterointimenetelmällä Matlab-ympäristössä tai sen voidaan antaa muodostua kehitetyn laskentalohkon avulla. Koodikirja on tallennettu konfiguraatiotiedostoon ja sen rakennetta on havainnollistettu kuvassa 14.

	Signaali 1	Signaali 2	Signaali 3	...	Signaali n	Etäisyys
Näytevektori						
Ajotilanne 1						
Ajotilanne 2						
Ajotilanne 3						
⋮						
Ajotilanne k						

Kuva 14 Koodikirjan rakenne. Eri ajotilanteille on määritelty ajotilannetta kuvaava koodivektori, mihin tietyllä hetkellä automaatiojärjestelmästä tullutta näytevektoria verrataan.

Tietyllä ajanhetkellä prosessin automaatiojärjestelmästä saaduista näytteistä muodostetaan näytevektori, joka sisältää tarkkailtavien signaalien arvot. Näytevektoria verrataan kaikkiin koodivektoreihin laskemalla koodivektorin ja näytevektorin välinen euklidinen etäisyys.

Ajotilanteeksi määrätään se, jota vastaavan koodivektorin etäisyys näytevektoriin on pienin. Näytevektori tunnistetaan aina johonkin ajotilanteeseen kuuluvaksi, joten etäisyys kertoo tunnistuksen luotettavuudesta. Tunnistettu ajotilanne ja etäisyys tallennetaan DNAhistorian-tietokantaan konfiguraatiotietokannassa annettuun osoitteeseen. Tämän jälkeen ajotilannetieto on muiden ylemmän tason sovellusten käytössä.

Kun näytevektori on yhdistetty johonkin ajotilanteeseen, tämän koodivektoria voidaan siirtää lähemmäksi näytevektoria laskemalla uusi sijainti rekursiivisella klusterointialgoritmilla. Tämä aiheuttaa sen, että koodivektorit hakeutuvat ajotilanteita vastaavien näytteiden läheisyyteen. Koodikirja voidaan määrätä myös kiinteäksi, jolloin koodivektoreiden sijaintia ei päivitetä.

Konfiguraatiotietokantaan voidaan tallentaa useita yksikköprosessikokonaisuuksia, jolloin laskentaa voidaan tehdä useille yksikköprosesseille tai laitteille samassa tietokantapalvelimessa.

4.4 Klusterointimenetelmien soveltuvuus tietokantaympäristöön

Tietokantaympäristö asettaa koodattavalle algoritmille seuraavanlaisia vaatimuksia:

- Algoritmi ei saa sisältää matriisioperaatioita, koska se pitää pystyä toteuttamaan SQL-skriptikielellä
- Algoritmin pitää olla rekursiivinen
- Klusteroinnin tulos pitäisi olla riippumaton koodivektoreiden alkuarvoista
- Algoritmin pitäisi olla laskennallisesti mahdollisimman kevyt suorittaa
- Käyttöönoton helpottamiseksi algoritmin pitäisi sisältää mahdollisimman vähän parametreja

Luvussa 3 esitellyistä algoritmeista mikään ei toteuta kaikkia vaatimuksia. Fuzzy c-means, Expectation Maximization ja K-means ovat batch-tyyppisiä algoritmeja, joten ne eivät tule kysymykseen.

Neural Gas on vähiten alkuarvoista riippuva rekursiivisesti suoritettava algoritmi, mutta "Neighbourhood ranking"-operaatio on raskas ja vaikea toteuttaa. Rekursiivinen K-means on kevyt ja yksinkertainen toteuttaa, mutta sen tulos riippuu alkuarvoista. SOM-algoritmi on myös rekursiivinen, mutta senkin tulos riippuu alkuarvoista ja algoritmissa tarvittavat naapuruussuhteet ovat käytännössä mahdottomia toteuttaa SQL kielellä.

Koska mikään tutkituista algoritmeista ei suoraan toteuta asetettuja vaatimuksia, diplomityössä kehitettiin rekursiivinen online-laskentaan soveltuva algoritmi, joka on mahdollista toteuttaa tietokantaympäristöön.

Toteutettu algoritmi perustuu rekursiiviseen K-Means klusterointialgoritmiin. Tässä toteutuksessa algoritmia on kuitenkin muokattu alkuarvoriippuvuuden vähentämiseksi. Algoritmiin on lisätty unohduskerroin, jolla pyritään parantamaan luokittelun adaptiivisuutta ja nopeuttamaan itseorganisoitumista. Unohduskerroin on välttämätön, koska online-algoritmissa tarvittava osumien lukumäärä kasvaisi muuten äärettömäksi.

Jokaiselle klusterille lasketaan klusteriin kuuluvien näytteiden lukumäärän lisäksi klusterin kokoa. Osumien lukumäärän ja klusterin koon avulla sellainen klusteri, johon ei ole tullut osumia pitkään aikaan voidaan unohtaa ja sijoittaa uudestaan isokokoisen klusterin läheisyyteen, jolloin klusteroinnin tulos paranee.

4.4.1 Käytetyn algoritmin matemaattinen kuvaus

Olkoon näyte hetkellä t vektori $x(t)$. Klusterilla Γ tarkoitetaan olioa, jolle on määritelty koodivektori v, osumien lukumäärä h ja klusterin koko d. Nämä klusterit muodostavat koodikirjan V. Koodikirja on siis k x p –matriisi, jossa p on signaalien lukumäärä ja k on klustereiden lukumäärä. Lisäksi määritellään unohduskerroin a välille [0,1]. Algoritmin suoritus etenee seuraavasti:

1. Alustetaan algoritmi antamalla koodivektoreille v alkuarvot

2. Luokitellaan näytevektori $x(t)$ lähimpään klusteriin Γ^c siten, että

$$x(t) \in \Gamma^c \text{ jos } \left\| x(t) - v^c \right\| = \min_j \left\{ \left\| x(t) - v^j \right\| \right\} \ \forall\, j = 1...k \tag{4.1}$$

3. Siirretään koodivektoria v^c kohti havaintoa:

$$v^c(t+1) = \frac{h_c(t)\, v^c(t) + x(t)}{h_c(t) + 1} \tag{4.2}$$

4. Päivitetään klusterin koko d.

5. Päivitetään osumien lukumäärä

$$h(t+1) = a h(t) \ \forall \Gamma \tag{4.3}$$

$$h_c(t+1) = h_c(t) + 1 \tag{4.4}$$

6. Jos jollekin klusterille pätee $h < 1$, klusterin koodivektoriksi kopioidaan sen klusterin koodivektori, jolle osumien lukumäärästä h ja klusterin koosta d riippuva kriteeri f on suurin

$$\Gamma^{max} = \max_j \left\{ f(h_j, d_j) \right\} \forall j = 1...k \tag{4.5}$$

Siirrettävälle ja alkuperäiselle klusterille Γ^{max} :

$$h(t+1) = h_{\max}(t) \ , \ d(t+1) = \frac{d_{\max}(t)}{2} \tag{4.6}$$

7. $t = t+1$, Palataan kohtaan 2. (4.7)

Vaiheessa 6 huonon alkuarvon saanut klusteri unohdetaan ja sen avulla kriteerin f maksimoiva klusteri jaetaan kahteen osaan. Tämän operaation ansiosta alkuarvoriippuvuus vähenee ja klusteroinnin tulos paranee.

4.4.2 Algoritmin ominaisuuksia

Algoritmi vaatii parametrina klustereiden lukumäärän lisäksi unohduskertoimen a ja funktion f. Mikäli unohduskerroin on liian suuri algoritmin antama tulos vastaa K-means algoritmia ja se hakeutuu kvantsointivirheen lokaaliin minimikohtaan, koska vaihetta 6 ei koskaan suoriteta. Mikäli unohduskerroin on liian pieni, klusterit unohtuvat usein ja siirtyvät kuvaamaan viimeisimpiä arvoja.

Funktiolla f voidaan painottaa joko osumien lukumäärää tai klusterin kokoa. Datan kvantisoinnissa halutaan minimoida kvantisointivirhe E eli kuvata tarkasti se osa näyteavaruudesta jossa näytteitä on paljon. Tällöin kriteerinä kannattaa käyttää osumien lukumäärää,

$$f(h,d)=h \tag{4.8}$$

Näytteiden klusteroinnissa näytejoukosta halutaan löytää ryhmiä, jolloin ei ole mitään mieltä sijoittaa useita koodivektoreita samaan ryhmään, vaikka siihen kuuluisikin paljon näytteitä. Klusterointisovelluksessa kannattaa kriteeriin sisällyttää klusterin koko d, jolloin tarkoituksena on minimoida klustereiden koko,

$$f(h,d)=hd^2 \tag{4.9}$$

Vaikka algoritmi on kehitetty online-laskentaan, sillä voidaan klusteroida myös olemassaolevia näytejoukkoja offline laskentana. Klusteroitaessa olemassa olevaa näytejoukkoa, kannattaa unohduskerroin mitoittaa siten, että osumien lukumäärän putoaminen alkuarvosta h_0 unohdusrajalle vaatii ainakin yhden kokonaisen iteraation. Toisen iteraation alussa ne koodivektorit joihin ei tullut yhtään osumaa ensimmäisellä iteraatiolla siirretään vaiheessa 6 sellaisten koodivektoreiden läheisyyteen, josta näytteitä on löytynyt.

Unohduskertoimen arvo voidaan määrittää seuraavasti. Olkoon näytteiden lukumäärä n ja

klustereiden lukumäärä k.

$$h_0 = \frac{n}{k} \tag{4.10}$$

$$h(i+1) = ah(i) \tag{4.11}$$

$$h(n) = a^n h_0 = 1 \rightarrow a^n = \frac{1}{h_0} \rightarrow n\ln(a) = \ln\left(\frac{1}{h_0}\right) \rightarrow a = e^{\frac{-\ln(h_0)}{n}} \tag{4.12}$$

$$a = e^{\frac{-\ln(h_0)}{n}} = e^{\frac{-\ln\left(\frac{n}{k}\right)}{n}} = \left[e^{-\ln\left(\frac{n}{k}\right)}\right]^{\frac{1}{n}} = \left[\frac{1}{e^{\ln\left(\frac{n}{k}\right)}}\right]^{\frac{1}{n}} = \left(\frac{k}{n}\right)^{\frac{1}{n}} \tag{4.13}$$

Tällä unohduskertoimella klusteri unohdetaan jos ja vain jos siihen ei ensimmäisen iteraation aikana tule yhtään osumaa. Seuraavilla iteraatioilla klusteriin täytyy tulla iteraation aikana vähintään h_0 osumaa ettei sitä unohdettaisi. Jos näytteet jakautuvat tasan klustereiden välille, mitään klusteria ei unohdeta.

Tällä tavalla laskettuna unohduskertoimelle saadaan käytännössä alaraja. Unohduskerroin kannattaa mitoittaa hieman isommaksi, koska näytteet eivät käytännössä jakaudu tasaisesti eri klustereihin.

Online-sovelluksessa unohduskertoimen määritys ei ole yhtä selkeää. Unohduskerroin pitäisi suhteuttaa analysoitavan ilmiön dynamiikkaan. Mikäli näyteväliksi oletetaan 1 aikayksikkö, osumien lukumäärä noudattaa ensimmäisen kertaluvun dynamiikkaa ja unohduskertoimen ja aikavakion välillä on riippuvuus:

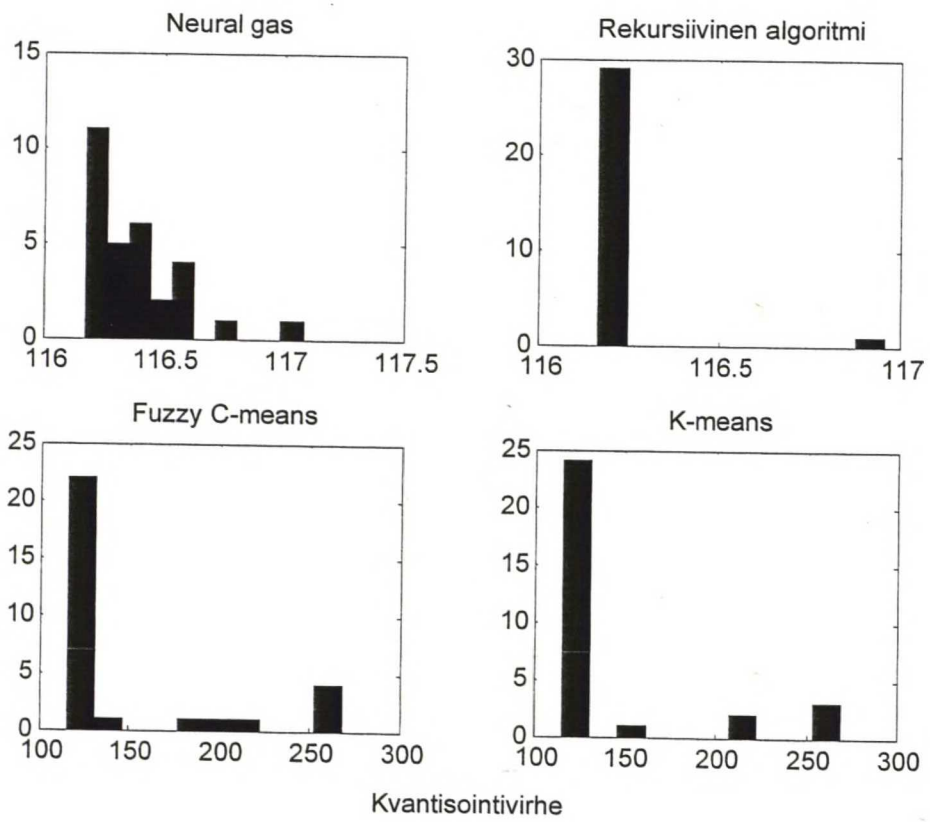
$$\tau \approx \frac{a}{1-a} \tag{4.14}$$

$$h(t) = h_0 e^{-(t/\tau)} \tag{4.15}$$

Osumien lukumäärän dynamiikka pitäisi olla järjestelmää hitaampi siten, että eri ajotilanteet toistuvat useaan kertaan aikana, joka kuluu klusterin unohtamiseen eli $h(t)$ alittaa arvon 1. Esimerkiksi eri paperilajeja ajetaan kerralla useita päiviä, joten muita lajeja kuvaavat klusterit eivät saa unohtua tällä välin. Unohduskerroin täytyy asettaa niin

suureksi, ettei näin pääse käymään. Toteutuksessa algoritmin vaihe 6 voidaan myös estää mikäli klusterit nimetään.

Algoritmin alkuarvoriippuvuutta verrattiin K-means, FCM ja Neural gas-algoritmeihin klusteroimalla luvussa 3 esitetty testidata 30 kertaa eri alkuarvoista. Jokaiselle yritykselle laskettiin kvantisointivirhe kaavalla 3.1. Kaikilla algoritmeilla klustereiden lukumäärä oli neljä. Koodivektoreiden alkuarvot valittiin satunnaisesti. Iteraatioiden määrä oli Neural Gas, FCM ja rekursiivisella online-algoritmilla neljä, mutta K-means algoritmin suoritusta jatkettiin lopetusehtoon asti. Eri algoritmien suorituskyky määritettiin laskemalla eri alkuarvoista saatujen kvantisointivirheiden jakaumat eri algoritmeille. Nämä on esitetty kuvassa 15.



Kuva 15 Testidatalle lasketun kvantisointivirheen histogrammi eri algoritmeille. Testidata klusteroitiin 30 kertaa eri algoritmeilla lähtien satunnaisista koodivektoreiden alkuarvoista. Työssä kehitetty rekursiivinen algoritmi päätyi yhtä yritystä lukuunottamatta samaan tulokseen.

4.5 Signaalien esikäsittely

Näytevektori muodostetaan DNAhistorian-tietokannassa olevista mittaus-, ohjaus- ja asetusarvosignaaleista joko valitsemalla tietokannasta valmiiksi löytyviä muuttujia tai muodostamalla uusia piirteitä jonkin funktion avulla. Esimerkiksi signaalien suhteet saattavat olla ajotilanteen kannalta merkittävämpiä kuin signaalien absoluuttiarvot. Käytännössä käytettävät muuttujat ratkaisevat toimiiko tunnistus halutulla tavalla vai ei. Uusien piirteiden muodostaminen on kuitenkin sovelluskohtainen operaatio, joten piirteiden muodostaminen täytyy tehdä järjestelmän ulkopuolella.

Järjestelmä on toteutettu siten, että piirteen muodostava SQL-lause voidaan kirjoittaa erilliseen tekstitiedostoon, johon viitataan konfiguraatiotietokannasta. Tällä tavalla voidaan muodostaa useasta tietokantasignaalista sovelluskohtainen tunnusluku yleiskäyttöisesti ilman, että varsinaista sovelluskoodia tarvitsee muokata.

Virheelliset tai puuttuvat näytteet vaikeuttavat itseorganisoitumista ja heikentävät koodikirjan laatua. Virheellisiä näytteitä syntyy kun tiedonsiirto automaatiojärjestelmästä tietokantaan estyy esimerkiksi yhteyskatkon takia. Virhetilanteessa tietokantaan tallentuu tilanteesta riippuen tyhjää, nolla, 'NaN' tai ei mitään, jolloin vanha arvo pysyy voimassa.

On-line algoritmissa näyte on kuitenkin vaikea todeta virheelliseksi, joten tässä toteutuksessa joudutaan olettamaan, että virheellisen datan määrä tulee olemaan erittäin pieni suhteessa laadukkaaseen dataan. Jonkin piirrevektorin alkion puuttuessa ei tunnistusta eikä päivitystä tehdä. Laskenta jatkuu itsestään kun piirrevektorin kaikki alkiot ovat taas luettavissa. Mikäli järjestelmä opetetaan historiadatan perusteella offline-opetuksena, vialliset näytteet on helppo poistaa opetusdatasta.

Skaalauksella pyritään saamaan eri mittayksiköissä olevat signaalit keskenään vertailukelpoisiksi. Tämä tehdään poistamalla signaalista pitkän ajan keskiarvo ja jakamalla se pitkän ajan keskihajonnalla, jolloin skaalatun signaalin varianssi tulee olemaan suurin piirtein yksi.

Tulosignaalille x tarvitaan siis kolme skaalausparametria: keskiarvo μ_x, keskihajonta σ_x ja painokerroin w. Keskiarvo ja keskihajonta voidaan estimoida opetusdatasta, mutta painokerroin vaatii asiantuntemusta. Näytteet skaalataan ennen klusterointia kaavalla 4.16. Nämä tunnusluvut määritellään jokaiselle tulosignaalille ja ne tallennetaan konfiguraatiotietokantaan.

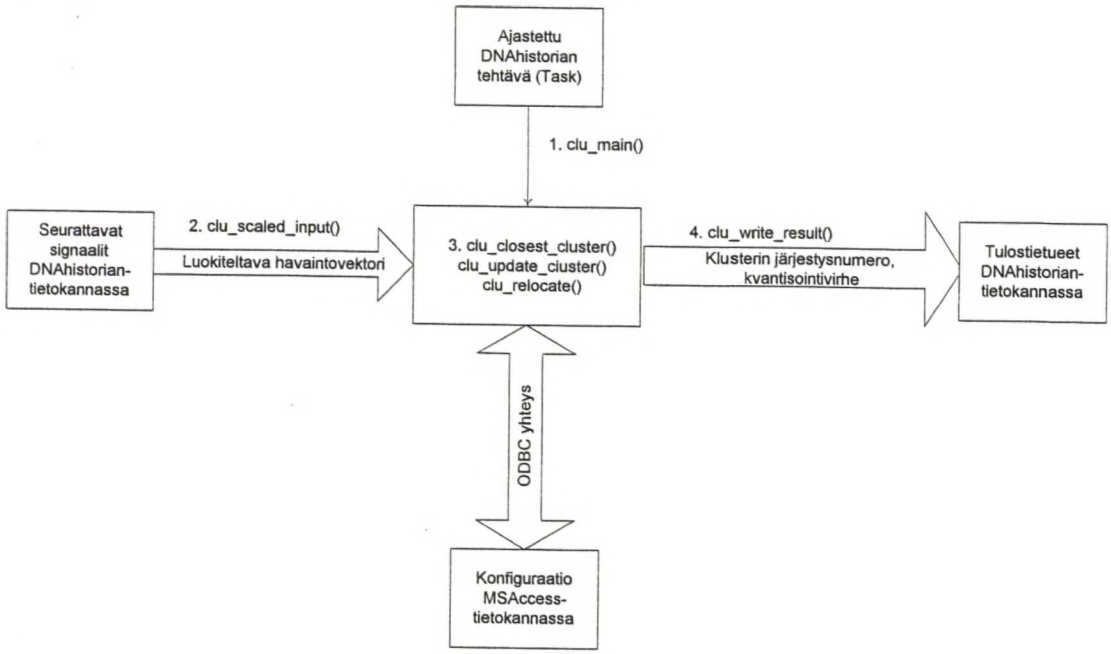
$$x_s(t) = \frac{w}{\sigma_x}(x(t) - \mu_x) \tag{4.16}$$

4.6 Toteutus tietokantaympäristöön

SQL-koodi koostuu yhdeksästä funktiosta. Lisäksi on yksi yleiskäyttöinen asennusohjelma, joka luo laskennan tuloksille omat DNAhistorian tietokantaosoitteet sekä asettaa pääproseduurin kutsun osaksi DNAhistorian-tehtävää. Kaikki lohkoon kuuluvat funktionimet ovat tyyppiä clu_xxx. SQL-koodin arkkitehtuuri on esitetty kuvassa 16.

SQL-koodiin kuuluvat funktiot ovat:

- FUNCTION *clu_main*(variable_id integer)
- PROCEDURE *clu_scaled_input*(variable_id integer)
- FUNCTION *clu_closest_cluster*(variable_id integer, min_distance out real)
- FUNCTION *clu_dist_input2cluster*(op_id integer)
- PROCEDURE *clu_next_location*(op_id integer)
- PROCEDURE *clu_read_location*(op_id integer)
- PROCEDURE *clu_relocate*(variable_id integer, op_to_move integer)
- PROCEDURE *clu_update_cluster*(variable_id integer, op_id integer, distance real)
- PROCEDURE *clu_write_result*(variable_id integer, op_id integer, distance real)
- PROCEDURE *clu_create_op*(pif char(15), rep char(15), etr char(15), plant_area char(13), oper_point_var_id int, t_update int)



Kuva 16 Tietokantaympäristöön toteutetun klusterointikoodin arkkitehtuuri.

DNAhistorian tehtäväsäie kutsuu määrävälein pääfunktiota *clu_main*, joka luo ensin kaksi taulukkoa, joista toisessa säilytetään näytevektoria ja toisessa yhden klusterin koodivektoria. Nämä ovat globaaleja muuttujia eli ne näkyvät kaikille pääohjelman kutsumille funktioille.

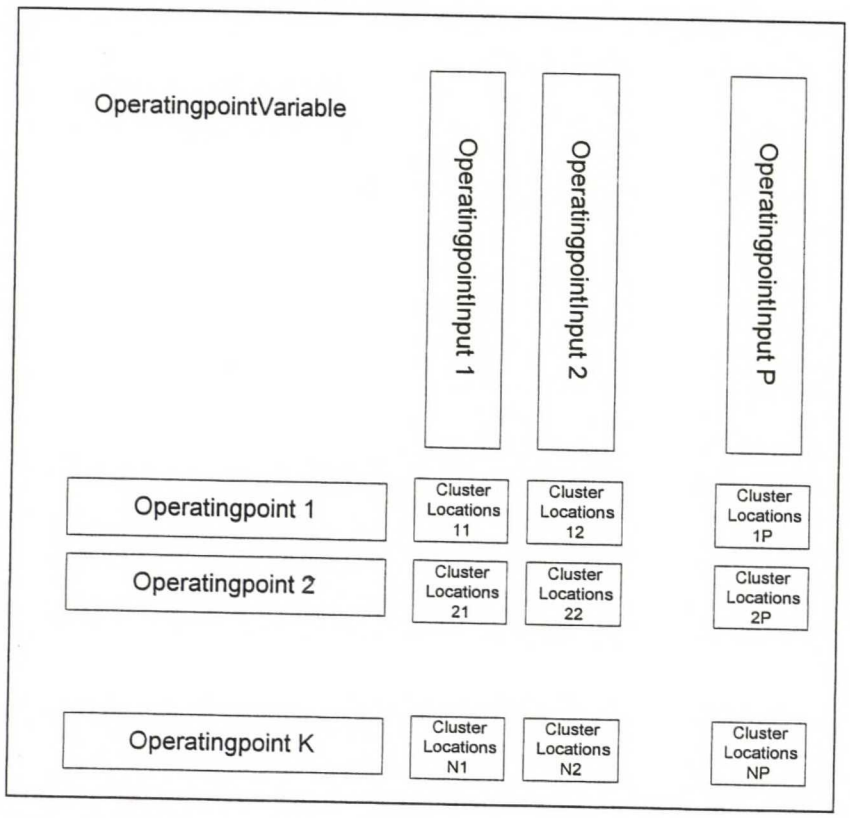
Näytevektori luetaan *clu_scaled_input* funktiolla. Tämän jälkeen *clu_closest_cluster* tutkii mikä konfiguraatiotiedostoon tallennetuista koodivektoreista on lähinnä näytevektoria. Tämä on monimutkaisin funktio ja se koostuu kahdesta alifunktiosta: *clu_next_cluster* ja *clu_dist_input2cluster*. Edellinen lukee klusterin koodivektorin konfiguraatiotiedostosta muistiin ja jälkimmäinen laskee näytevektorin ja koodivektorin välisen etäisyyden neliön.

Mikäli koodikirjaa on lupa päivittää, lähimmän klusterin koodivektoria päivitetään *clu_update_cluster*-funktiolla. Tämän jälkeen tutkitaan onko jonkin klusterin näytemäärä painunut siirtorajan alle ja jos on se siirretään *clu_relocate*-funktiolla suurimman klusterin läheisyyteen ellei operaatiota ole kielletty.

Kun lähin klusteri tunnetaan, sitä vastaava koodinumero kirjoitetaan annettuun osoitteeseen *clu_write_result*-funktiolla. Tämän jälkeen ajotilannetieto on tallennettu määrättyyn positioon, josta sitä voidaan hyödyntää missä tahansa sovelluksessa.

4.7 Konfiguraatiotietokanta

Konfiguraatiotietokantaan on tallennettu järjestelmän vaatimat tiedot. Erillisen konfiguraatiotietokannan tarkoitus on tehdä laskentalohkosta mahdollisimman yleiskäyttöinen. Konfiguraatiotietokanta on Microsoft Access tyyppinen relaatiotietokanta ja se koostuu neljästä taulusta: OperatingPointVariable, OperatingPointInput, OperatingPoint ja ClusterLocations. Tietokannan rakennetta on havainnollistettu kuvassa 17. Laskentalohko lukee jokaisella kierroksella konfiguraatiotietokantaa ja päivittää tauluja OperatingPoint ja ClusterLocations. Konfiguraatiotietokantaan voidaan tallentaa tiedot useiden yksikköprosessien ajotilanteen selvittämiseksi.



Kuva 17 Konfiguraatiotietokannan rakenne. Tietokannan rakenne vastaa täysin klusterointialgoritmin koodikirjan rakennetta.

OperatingPointVariable-tauluun määritellään yhden yksikköprosessin ajotilanteen tunnistamiseen liittyviä asioita:

- OperatingPointVariableID (Integer), Yksikköprosessin tunnistenumero.
- Position (String), DNAhistorian-tietokannan positiotunnus, johon laskennan tuloksena syntyvä ajotilanteen koodinumero tallennetaan.
- DistancePosition (String), DNAhistorian-tietokannan positiotunnus, johon laskennan tuloksena syntyvä tunnistuksen luotettavuutta kuvaava kvantisointivirhe tallennetaan.
- ForgetFactor (Float), Rekursiivisen keskiarvon laskennassa käytettävä unohduskerroin
- Updating (Boolean), Määrää päivitetäänkö koodikirjaa vai ei.
- RelocationLock (Boolean), Määrää voidaanko algoritmin vaihe 6 suorittaa

OperatingPointInput-tauluun määritellään signaalit, joiden perusteella ajotilanteen tunnistus tapahtuu.

- OperatingPointInputID (Integer), Mittaussignaalin tunnistenumero.
- OperatingPointVariableID (Integer), Se yksikköprosessi, johon mittaussignaali kuuluu.
- Mean (Float), Signaalin keskiarvo, joka vähennetään signaalin arvosta ennen laskentaa, jotta signaali saadaan nollakeskiarvoiseksi.
- STD (Float), Signaalin keskihajonta, jolla signaalin arvo jaetaan ennen laskentaa, jotta signaalin varianssi saadaan ykköseksi.
- Weight (Float), Signaalin painokerroin, jolla skaalattu signaali kerrotaan ennen laskentaa.
- RecordName (String), DNAhistorian-tietokannan positiotunnus, josta signaalin arvo luetaan.
- Description (String), Signaalille mahdollisesti annettava kuvaus.
- FeatureFunction (String), Piirteen muodostuksessa käytettävän funktion sijainti, mikäli signaali ei ole suoraan luettavissa jostain tietokannan positiosta.
- Locked (Boolean), Määrää päivitetäänkö koodivektoreiden tätä signaalia vastaavaa alkioita.

OperatingPoint-tauluun määritellään yksikköprosessin ajotilanteiden nimet ja kuvaukset.

- OperatingPointID (Integer), Yksittäisen ajotilanteen tunnistenumero.
- OperatingPointVariableID (Integer), Sen yksikköprosessin tunnistenumero,

johon ajotilanne liittyy.

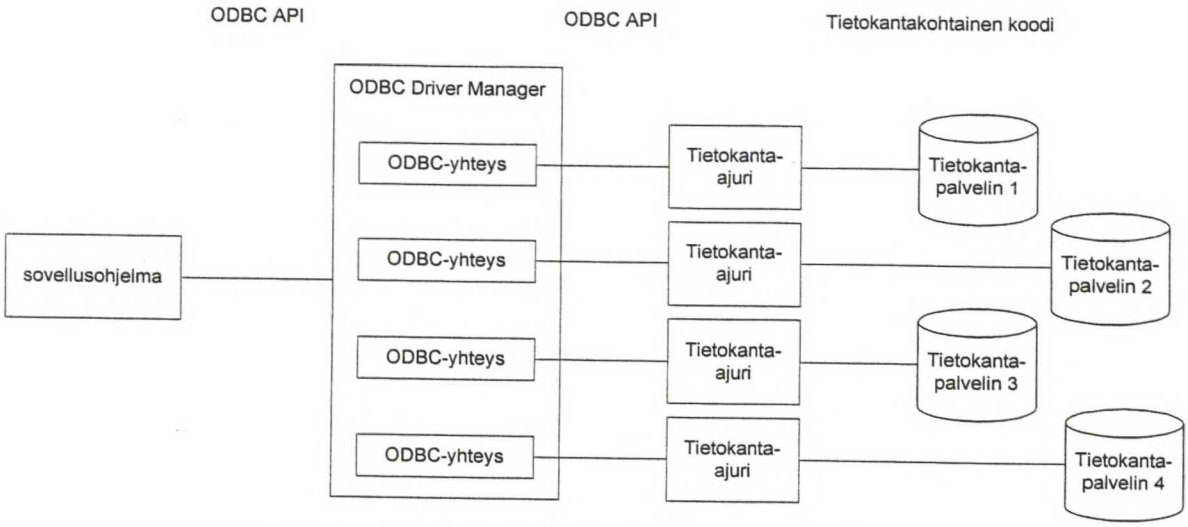
- Name (String), ajotilanteen nimi.
- Description (String), ajotilanteen sanallinen kuvaus.
- Value (Integer), Ajotilannetta kuvaava numero, joka tallennetaan DNAhistorian-tietokantaan.
- NumberOfHits (Float), Luku, joka kertoo montako kertaa ajotilanne on tunnistettu.
- AvgDistance (Float), Luku, joka kertoo ajotilannetta vastaavan koodivektorin keskimääräisen etäisyyden klusteriin kuuluviin näytteisiin.
- Locked (Boolean), Määrää onko tämän klusterin koodivektoria lupa päivittää tai siirtää.

ClusterLocations-tauluun määritellään ajotilanteita vastaavat koodivektorit skaalattuna.

- ID (Integer), on yhden klusterin tunnistenumero.
- OperatingPointVariableID (Integer), Viittaus osaprosessiin, johon tämä klusteri kuuluu.
- OperatingPointID (Integer), Viittaus siihen ajoilanteeseen, jota klusteri kuvaa.
- OperatingPointInputID (Integer), Signaalin tunnistenumero.
- ScaledValue (Float), Signaalin skaalattu arvo. Klusterin sijaintitieto on tallennettu muodossa OperatingPointInputID = ScaledValue.

4,8 Open Database Connectivity (ODBC)

ODBC on Microsoft Corporationin kehittämä spesifikaatio ohjelmointirajapinnalle, jonka avulla eri sovellusohjelmien on mahdollista päästä käsiksi useissa eri valmistajien tietokantajärjestelmissä olevaan dataan. ODBC:n avulla sovellusohjelma on mahdollista kehittää riippumattoksi tietokantajärjestelmästä. ODBC-yhteys käsittää verkkoyhteyden tietokantapalvelimen ja työaseman välillä, tietokannan kanssa yhteensopivan ODBC-ajurin ja tietokantaa kuvaavan ODBC-nimen. Sovellusohjelman tarvitsee tietää vain tietokannan ODBC-nimi ja tietokannan rakenne tiedon hakua varten. Tieto tietokannan rakenteesta tarvitaan, koska tietoa haetaan SQL-lauseilla, joiden kirjoittamiseen tarvitaan taulu- ja sarakenimiä sekä tietotyyppejä. ODBC-arkkitehtuuria on havainnollistettu kuvassa 18.



Kuva 18 Sovellusohjelma voi hakea ODBC:n avulla tietoa useista eri valmistajien tietokannoista.

ODBC-rajapinta määrittelee millä tavalla yhteys sovellusohjelmasta ODBC Driver Manager-sovellukseen tapahtuu ja miten tietokantakohtaisen ajurin tulisi toimia, jotta kommunikointi onnistuisi.

Vaikka Microsoft onkin toteuteuttanut ODBC:n Windows maailmassa, ODBC ei ole millään tavalla käyttöjärjestelmäriippuvainen, vaan ODBC toteuksia on tehty myös Macintosh ja UNIX-ympäristöissä [Mic98].

Käytännössä melkein kaikki tietokantajärjestelmät tukevat ODBC-rajapintaa ja sitä käytetään erittäin laajasti tietokantasovelluksissa. Microsoft Accessin ODBC-ajuri on osa Windows-käyttöjärjestelmää ja siksi ohjelmistojen konfigurointimahdollisuus on käytännöllistä toteuttaa Microsoft Access:in ja ODBC-yhteyksien avulla. ODBC yhteydessä tieto liikkuu monen rajapinnan läpi, jolloin yhteys on huomattavasti hitaampi kuin suoraan tietokanta-ajuria käytettäessä.

Ajotilanteen tunnistusjärjestelmässä käytetään ODBC-yhteyttä DNAhistorian- ja konfiguraatiotietokannan välillä, joten DNAhistorian palvelimen DriverManager-asetuksiin on lisättävä ODBC-linkki konfiguraatiotietokantaan.

4.9 Käyttöönotto ja Matlab-konfiguraattori

Matlab-ympäristöön toteutettiin funktiokirjasto, jonka avulla ajotilanteen tunnistusjärjestelmä voidaan ottaa käyttöön prosessin historiadatan tai prosessituntemuksen avulla. Kirjasto koostuu datanhaku-, datankäsittely- ja käyttöönottofunktioista. Kirjaston sisällysluettelo on esitetty liitteessä 2.

Prosessidataa voidaan hakea ODBC-yhteyden avulla DNAhistorian tietokannasta suoraan Matlab-ympäristöön, jossa sitä on helppo käsitellä ja visualisoida. Prosessidataa tarvitaan tunnistusjärjestelmän opetukseen ja skaalausparametrien määritykseen. Kun ajotilanteen tunnistamiseen vaadittavat tiedot ovat selvillä ja tunnistusjärjestelmä toimii halutulla tavalla, voidaan konfigurointitiedot siirtää Matlab-ympäristöstä konfigurointitietokantaan tätä varten toteutetulla ohjelmistokomponentilla.

4.9.1 ODBC-tietokantarajapinta Matlabissa

Tietokantaominaisuuksia tarvittiin toisaalta prosessidatan siirtämiseksi DNAhistorian tietokannasta Matlab-ympäristöön ja toisaalta koodivektoreiden siirtämiseksi konfiguraatiotietokannasta takaisin Matlab-ympäristöön, jotta laskennan tuloksia voidaan seurata.

Matlabiin on saatavilla tietokantayhteyksiä varten useita funktiokirjastoja. Työssä käytettiin vapaasti levitettävää Dbtool–nimistä kirjastoa, joka osoittautui varmatoimiseksi ja helppokäyttöiseksi. Dbtool sisältää funktiot ODBC-yhteyden avaamista, SQL-lauseen suoritusta ja yhteyden sulkemista varten. Datan haku toimii samalla periaatteella kuin ohjelmointikielissä yleensäkin [He02].

Yhteyden avaamisessa tarvitsee tietää ainoastaan tietokannan ODBC-nimi. Yhteyden avaaminen tuottaa yhteysolion, SQL-lauseen suoritus palauttaa onnistuessaan 'kursorin' haetun datajoukon ensimmäiselle riville. Riviltä luetaan halutut arvot talteen ja kursoria siirretään silmukassa seuraavalle riville. Matlabissa halutaan käsitellä matriisimuotoista dataa, jolloin datamatriisin muodostaminen vaatii ylimääräistä koodia. Kun viimeinenkin rivi on luettu yhteydet suljetaan. Tämä on ohjelmoitu yleiskäyttöiseksi funktioksi työssä toteutettuun funktiokirjastoon.

4.9.2 Tunnistusjärjestelmän opetus

Funktiokirjastossa on toteutettu funktio, joka skaalaa opetusdatan edellä esitetyllä tavalla ja suorittaa klusteroinnin annetulle datalle työssä kehitetyllä algoritmilla. Funktio palauttaa löydettyjen klustereiden koodivektorit sekä tiedon jokaista näytettä lähimmästä klusterista. Matlab-ympäristöön on myös saatavilla Teknillisessä korkeakoulussa informaatiotekniikan laboratoriossa kehitetty SOMToolbox-funktiokirjasto, jonka ominaisuuksia voidaan käyttää datan käsittelyyn, järjestelmän opetukseen sekä tulosten visualisointiin. SOM Toolbox:iin on toteutettu itseorganisoituvan kartan lisäksi Neural Gas ja K-Means-algoritmit. Lisäksi valvottuun opetukseen on toteutettu algoritmeja [Alh00].

SOMToolbox-kirjaston klusterointialgoritmien toteutukset vaativat parametrina skaalatun opetusdatamatriisin ja palauttavat löydetyn koodikirjan, joten ne ovat täysin yhteensopivia tässä työssä toteutetun järjestelmän kanssa.

Itseorganisoituvan kartan koodivektorit voidaan tallettaa sovitussa järjestyksessä konfiguraatiotietokantaan, jolloin yksikköprosessin tilan trajektoria kaksiulotteisella kartalla pystytään seuraamaan.

4.9.3 Konfiguraation tallennus konfiguraatiotietokantaan

Koodikirja saattaa koostua sadoista koodivektoreista, joilla on useita alkioita, joten tämä vaihe on liian työläs ja virhealtis käsin kirjoitettavaksi. Tästä johtuen työssä kehitettiin Matlab-funktio, jolla iso koodikirja saadaan automaattisesti siirrettyä Matlab-ympäristöstä konfiguraatiotietokantaan. Siirrossa hyödynnetään ODBC-yhteyttä, joten konfiguraatiotietokanta täytyy lisätä myös sen tietokoneen ODBC-konfiguraatioon, jolla konfigurointia tehdään.

Funktiota käytettäessä tarvitsee tietää:

- konfiguraatiotietokannan ODBC-osoite (merkkijono)
- yksikköprosessin nimi (merkkijono)
- unohduskerroin (liukuluku välillä [0,1])
- itseorganisoituminen käytössä/ei käytössä (1/0)
- seurattavien signaalien tietokantaosoitteet (cell array)
- signaalien keskiarvot (1 x p vektori)
- signaalien keskihajonnat (1 x p vektori)
- signaalien keskinäiset painokertoimet (1 x p vektori)

- koodikirja. (k x p matriisi)

Muuttujien osalta on tärkeää, että signaalien tietokantaosoitteet ovat samassa järjestyksessä kun niiden keskiarvot, keskihajonnat ja painokertoimet, jotta signaalit skaalattaisiin käyttäen oikeita parametreja.

Matlab funktio välittää tiedot Java-sovellukselle, joka muodostaa tietojen perusteella konfiguraatiotietokantaa vastaavan oliorakenteen. Tämän jälkeen Java-sovellus tallentaa JDBC-ODBC rajapintaa hyväksi käyttäen oliorakenteen konfiguraatiotietokantaan.

Ajotilanteiden nimet ja koodinumerot voidaan vaihtaa käyttäen Microsoft Access-ohjelmaa. Java-sovellus palauttaa laskentalohkon ID-numeron, jota tarvitaan käyttöönoton seuraavassa vaiheessa. Tämän jälkeen konfiguraatiotiedosto on valmis ja se voidaan kopioida tietokantapalvelimelle.

Lopuksi laskenta täytyy asettaa osaksi DNAhistorian tehtävää. Tässä vaiheessa tarvitaan seuraavat tiedot:

- DNAhistorian tietokannan ODBC-osoite (merkkijono)
- Sen DNAhistorian tehtäväsäikeen nimi, jonka halutaan suorittavan laskentaa (merkkijono)
- Yksikköprosessin nimi, josta generoidaan taginimet tulostietueille (merkkijono)
- Laskentalohkon ID-numero, jotta laskentalohko käyttäisi oikeaa konfiguraatiota. (clu_create_confdb-funktion palauttama arvo)
- Laskentaväli minuutteina

4.10 Liityntä muihin suorituskykyä seuraaviin sovelluksiin

Ajotilanteen tunnistusjärjestelmän ulostulona on ajotilannetta vastaava koodinumero sekä kvantisointivirhe. Molemmat tallentuvat DNAhistorian tietokantaan samalla tavalla kuin muutkin signaalit. Ajotilannetietoa on mahdollista käyttää kriteerinä datanhaun yhteydessä. Esimerkiksi säädön suorituskyvyn seurantaohjelmistot kykenenevät tekemään tällä tavalla säätöpiirien suorituskykyraportteja eri ajotilanteista, jolloin voidaan tutkia onko ajotilanteella vaikutusta säätöjen toimintaan.

Toinen mahdollisuus on käyttää kriteerinä kvantisointivirhettä. Prosessin tilan voimakkaat muutokset näkyvät piikkeinä kvantisointivirheessä. Rajoittamalla datahaku näiden piikkien ympärille voidaan analysoida säädinten toimintaa muutostilanteissa. Normaalit tuotantotilanteet taas ilmenevät alhaisena kvantisointivirheen arvona, jolloin datahaku voidaan rajoittaa ainoastaan normaalitilanteisiin.

Kaikki laskentalohkon tuottama informaatio on luettavissa joko DNAhistorian-tietokannasta tai konfiguraatiotietokannasta SQL-kyselyillä ODBC yhteyden avulla, joten lisäsovellusten kehittäminen on helppoa.

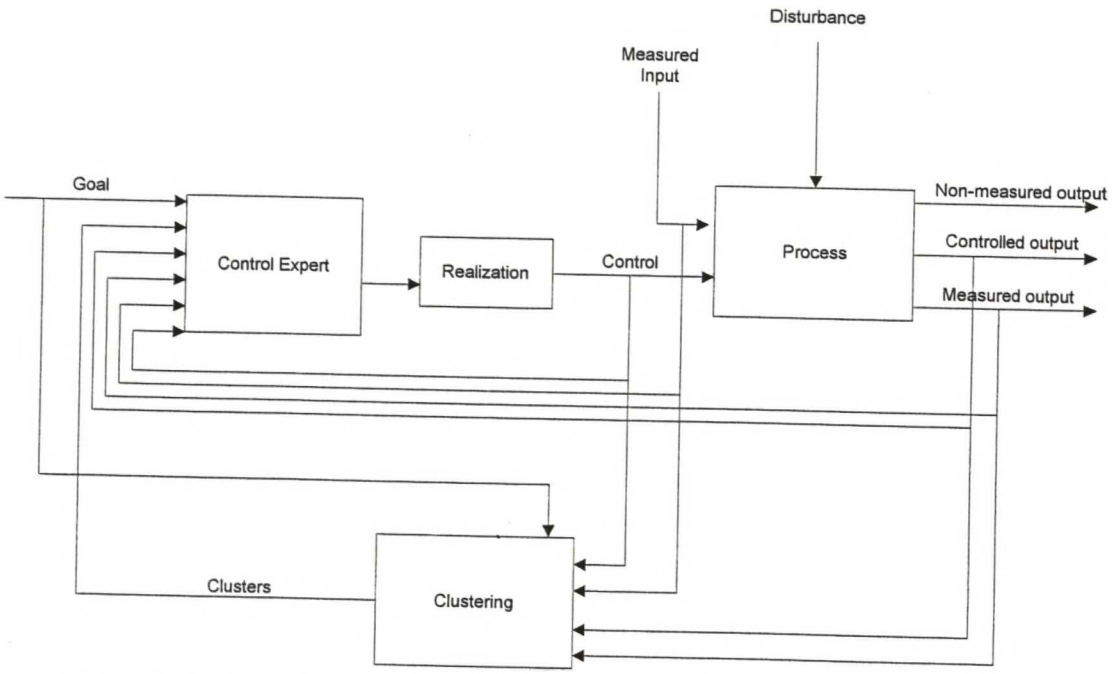
Työssä kehitettiinkin kvantisointivirheeseen perustuva diagnostiikkasovellus. Sovellus koostuu trendinäytöstä, johon piirretään klusteroinnin tuottama kvantisointivirhe ja tekstikonsolista. Muutos seurattavan prosessin toiminnassa havaitaan kvantisointivirheen kasvuna. Mikäli kvantisointivirhe kasvaa suureksi, diagnostiikka antaa tekstikonsoliin ilmoituksen poikkeavasta käyttäytymisestä. Vertaamalla poikkeavaa näytevektoria lähimpään koodivektoriin, voidaan poikkeava signaali myös tunnistaa ja kertoa mikä sen arvo normaalisti olisi muiden signaalien perusteella.

5 Sovelluskohteita

5.1 Adaptiivinen säätö

Epälineaarisuus aiheuttaa ongelmia prosessimallinnuksessa ja säätösuunnittelussa. Usein epälineaarisuus johtuu erilaisista prosessiolosuhteista, jotka vaikuttavat prosessin dynamiikkaan. Klusterianalyysillä voidaan etsiä olosuhteista riippuvia alueita, joiden sisällä prosessia voidaan kuvata lineaarisella mallilla. Epälineaarinen prosessi kuvataan siis usealla olosuhteiden mukaan identifioidulla lineaarisella mallilla.

Lineaaristen mallien perusteella voidaan suunnitella eri olosuhteisiin tarkoitettuja säätimiä. Online-klusterointiin perustuvaa adaptiivista säätöä on havainnollistettu kuvassa 19 [Yli94b].



Kuva 19 Klusterointiin perustuva adaptiivinen säätöratkaisu [Yli94b]. Säädössä huomioidaan klusteroinnin avulla useasta muuttujasta riippuva toimintapiste.

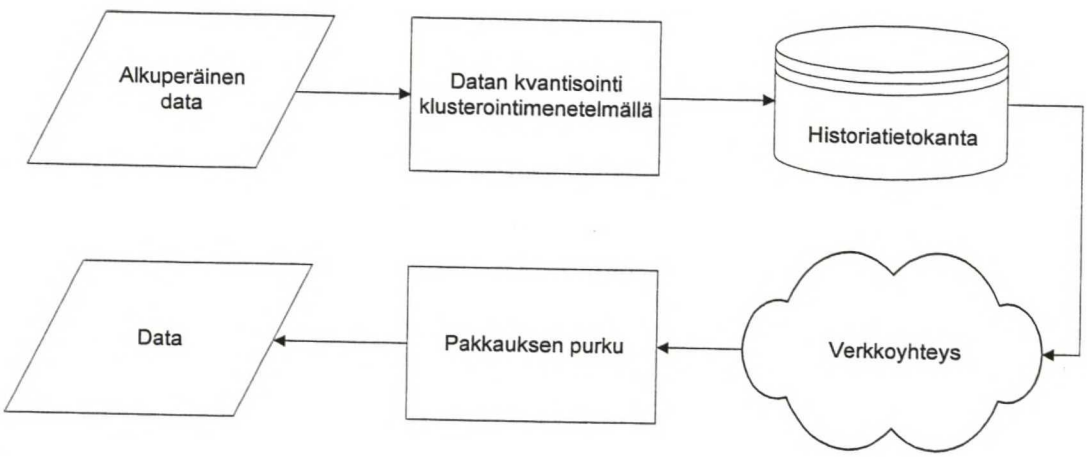
Prosessille identifioidaan Control Expert-lohkossa rekursiivisesti useaa lineaarista eri olosuhteita kuvaavaa prosessimallia, joiden perusteella voidaan laskea olosuhteisiin sopivat säätöparametrit. Clustering-lohko kertoo Control Expert-lohkolle voimassa olevat olosuhteet eli tiedon siitä, mitä mallia tämän hetkisillä mittauksilla pitäisi päivittää ja minkä mallin perusteella säätöparametrit pitäisi laskea. Realization-lohko on Control Expert-lohkon suunnitteleman säätimen toteutus.

Prosessimallin identifioinnissa voidaan käyttää rekursiivista pienimmän neliösumman menetelmää, jolloin mallit pysyvät ajantasalla. Vastaavasti klusteroinnissa päivitetään jatkuvasti koodikirjaa, jolloin se kykenee mukautumaan uusiin olosuhteisiin. Järjestelmä perustuu kuitenkin prosessin identifiointiin suljetussa järjestelmässä, mikä aiheuttaa rajoituksia.

Eri toimintapisteissä voidaan käyttää eri säätöparametreja, jolloin puhutaan Gain-scheduling-säädöstä. Klusterianalyysi paljastaa datasta ne toimintapisteet, joissa prosessi normaalisti toimii ja joihin säätimet kannattaa virittää. Hyvät säätöparametrit taulukoidaan toimintapistekohtaisesti. Gain-Scheduling säädössä säätöparametrit valitaan taulukosta toimintapisteen mukaan, mutta taulukoituja säätöparametreja ei automaattisesti muuteta [Åst95].

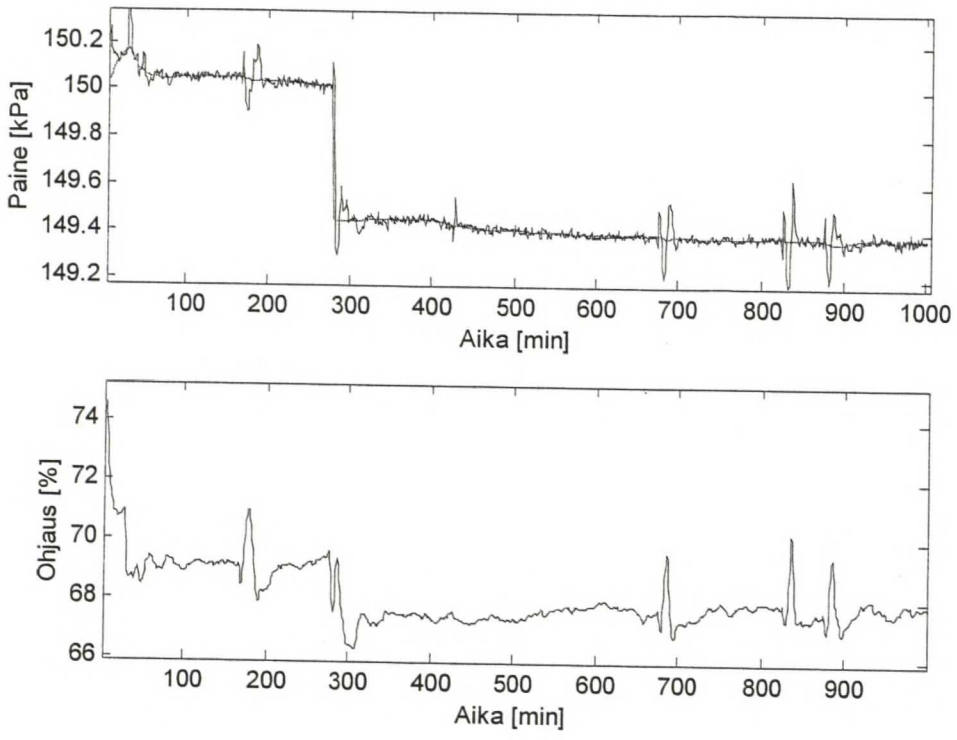
5.2 Datan pakkaus

Klusterointimenetelmillä voidaan kvantisoida datajoukko (n x p) koodikirjalla (k x p) ja yhdellä koodisignaalilla (n x 1). Lisäksi koodisignaali koostuu ainoastaan positiivisista kokonaisluvuista, jotka voidaan koodata vähemmällä määrällä bittejä, kuin alkuperäiset liukuluvut. Tämän jälkeen data on mahdollista rekonstruoida tarkkuudella, joka riippuu koodivektoreiden määrästä [Gra84]. Tehokasta datan pakkausta tarvitaan, kun dataa tallennetaan tietokantoihin tai lähetään verkkoyhteyden läpi (kuva 20).



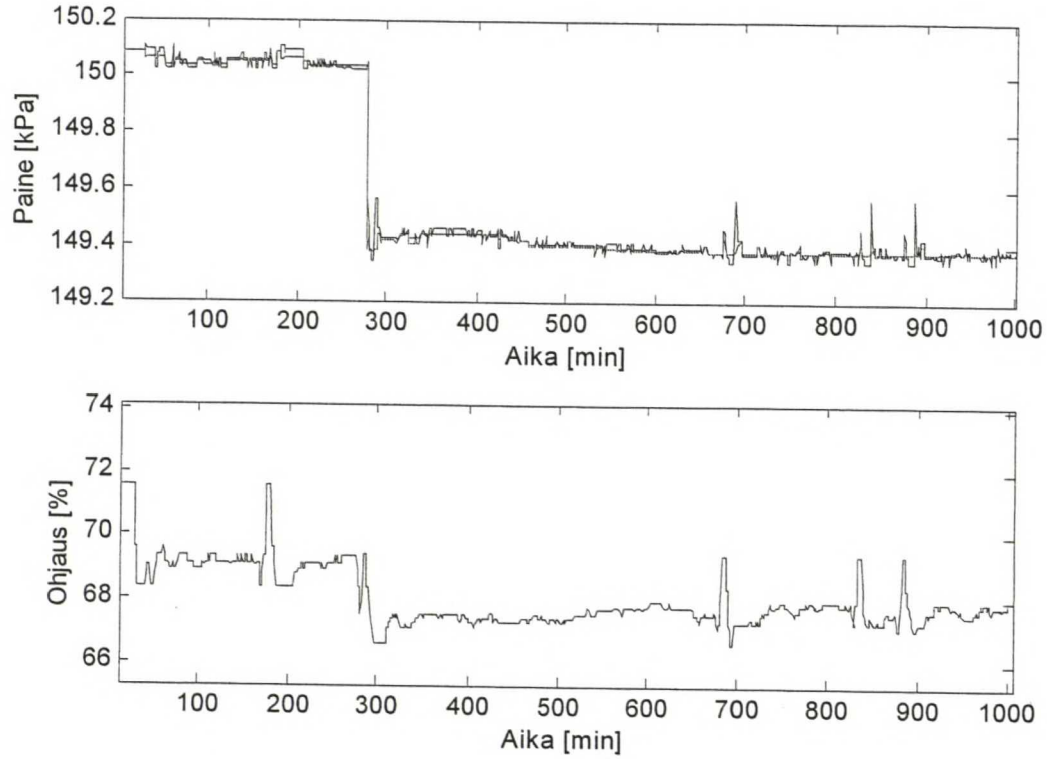
Kuva 20 Prosessidatan pakkaus klusterointimenetelmällä.

Seuraavassa esimerkissä pakataan erään paineensäätöpiirin asetusarvo-, mittaus- ja ohjaussignaali käyttäen työssä kehitettyä algoritmia. Dataa on kerätty noin 16 tunnin ajalta minuutin välein, näytteiden lukumäärä on 3x1000. Alkuperäinen data on esitetty kuvassa 21.



Kuva 21 Säätöpiiristä minuutin näytteenotolla kerätty asetusarvo-, ohjaus- ja mittaussignaali.

Klusterointialgoritmillä haettiin 40 koodivektoria sisältävä koodikirja, jolla data kvantisoidaan siten, että kvantisointivirhe minimoituu. Koodikirja sisältää 40x3=120 lukua. Koska pakkaus halutaan myös purkaa, tarvitaan koodikirjan lisäksi informaatio siitä, mikä koodivektori on tietyllä hetkellä voimassa. Tämä signaali sisältää myös 1000 lukua. Alkuperäinen 3000 lukua sisältävä data on nyt siis pakattu 1120 lukuun. Purettu data on esitetty kuvassa 22. Pakkausta voidaan edelleen tehostaa tallentamalla pelkästään muutokset ja niiden aikaleimat.



Kuva 22 Koodikirjan perusteella purettu data.

Pakkauksessa hävinnyttä informaation määrää voidaan arvioida laskemalla signaalille i selitetyn neliösumman ja kokonaisneliösumman välinen suhde eli selitysaste R_i^2,

$$R_i^2 = \frac{\sum_{t=0}^{n}(x_i(t)-\mu_i)^2 - \sum_{t=0}^{n}(x_i(t)-v_i(t))^2}{\sum_{t=0}^{n}(x_i(t)-\mu_i)^2} \tag{5.1}$$

jossa $x_i(t)$ on alkuperäinen signaali, μ_i on alkuperäisen sigaalin keskiarvo ja $v_i(t)$ on lähimmän koodivektorin signaalia x_i vastaava alkio.

Esimerkin tapauksessa selitysasteet ovat mittaussignaalille 0.981, ohjaussignaalille 0.939 ja asetusarvolle 0.995. Pakattu signaali vastaa melko hyvin alkuperäistä signaalia, joten klusterointimenetelmillä voidaan pakata dataa tehokkaasti.

5.3 Vikadiagnostiikka

Klusterointimenetelmien käyttö vikadiagnostiikassa perustuu oletukseen, että järjestelmän toimiessa normaalisti siitä saatavat mittaukset ja tunnusluvut muodostavat signaaliavaruuteen jonkinlaisen ominaiskuvion. Ominaiskuvio voi olla esimerkiksi yksittäinen datapilvi, useita datapilviä tai käyrä. Klusterianalyysin tuottamat koodivektorit approksimoivat tätä järjestelmän ominaiskuviota. Järjestelmän vikaantuessa järjestelmän ominaiskuvion oletetaan poikkeavan normaalista, jolloin keskimääräinen etäisyys näytevektorista lähimpään koodivektoriin kasvaa [Kas92].

Vian täytyy siis aiheuttaa selvä muutos ominaiskuviossa, jotta sen havaitseminen etäisyyden avulla olisi mahdollista. Tämän takia järjestelmästä täytyy mitata sellaisia tunnuslukuja, joissa mahdollinen vikaantuminen näkyy.

Periaatteessa vika voidaan klusterianalyysin avulla myös tunnistaa, jos vika aiheuttaa datan klusteroitumisen johonkin tiettyyn paikkaan avaruudessa. Viallisesta toiminnasta ei kuitenkaan yleensä ole saatavissa dataa, joten vikoja kuvaavien klusterikeskipisteiden määrittely ei myöskään ole yleensä pelkän datan perusteella mahdollista.

Kvantisointivirheeseen perustuva vian havaitseminen vaatii kvantisointivirhelle raja-arvon, jonka ylittämisestä seuraa vikailmoitus. Prosessin normaali siirtymä tilasta toiseen aiheuttaa piikin, koska prosessi siirtyy koodivektorin alueelta toisen koodivektorin alueelle. Näistä piikeistä ei kuitenkaan saa generoida hälytystä, koska kysymyksessä on normaali siirtymä. Raja-arvo voidaan antaa normaalista käyttäytymisestä saadun kokemuksen perusteella.

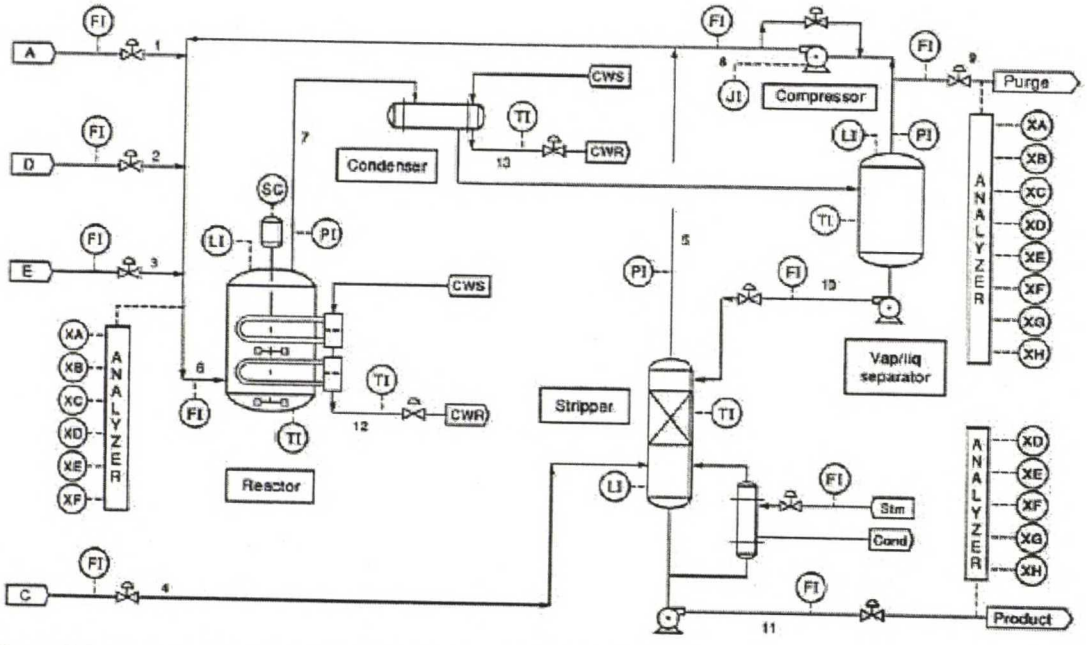
Tällaista lähestymistapaa on sovellettu paperikoneen telojen kunnonvalvonnassa [Rin00] ja lentokoneen suihkuturbiinimoottoreiden valvonnassa [Goe01]. Telojen kunnonvalvontasovelluksessa teloista mitattiin telojen nopeutta, momenttia, tehoa ja verkkojännitettä. Kvantisointivirheen muodostamiseen käytettiin itseorganisoituvaa karttaa, jolle syötettiin mittauksista muodostettuja aallokekertoimia. Järjestelmään syötettäviin signaaleihin generoitiin erilaisia testi-ilmiöitä, joista useimmat näkyivät selvästi kvantisointivirheessä.

Ongelmana tässä lähestymistavassa on kuitenkin se, että klusterianalyysi on suoritettava ainoastaan normaalitoiminnasta kerätylle datalle. Mikäli koodikirja kuvaa myös erilaisia vikatilanteita, kyseisiä vikoja ei havaita kvantisointivirheessä.

5.4 Case: Tennessee Eastman prosessi

Tennessee Eastman prosessi on Eastman Chemical Company:n kehittämä todellista tehdasprosessia kuvaava simuloitava prosessi. Prosessi on luonteeltaan epästabiili, epälineaarinen ja siinä on paljon tuntemattomia häiriöitä. Prosessia on yleisesti käytetty erilaisten säätö- ja vikadiagnostiikkamenetelmien kokeiluun.

Prosessi koostuu reaktorista, lauhduttimesta, kompressorista, separaattorista ja stripperistä. Lisäksi prosessissa on pumppuja ja säätöventtiilejä. Prosessissa virtaa kahdeksaa eri ainetta, jotka reagoivat keskenään tunnetulla tavalla. Prosessissa on 41 mitattavaa signaalia ja 12 ohjaussignaalia [Dow93]. Prosessin PI-kaavio on esitetty kuvassa 23.



Kuva 23 Tennessee Eastman prosessin PI-kaavio [Dow93].

Prosessisimulaattoriin on normaalitoiminnan lisäksi ohjelmoitu 21 eri vikatilannetta. Näistä kaikista on saatavissa dataa, jolla vikadiagnostiikan menetelmiä voi kokeilla. Eräs simuloitu vika on prosessissa olevan reaktorin jäähdytysveden virtauksen säätöventtiilissä esiintyvä takertelu. Venttiili ei liiku kitkan takia tasaisesti, vaan se generoi prosessiin värähtelyä [Bra01].

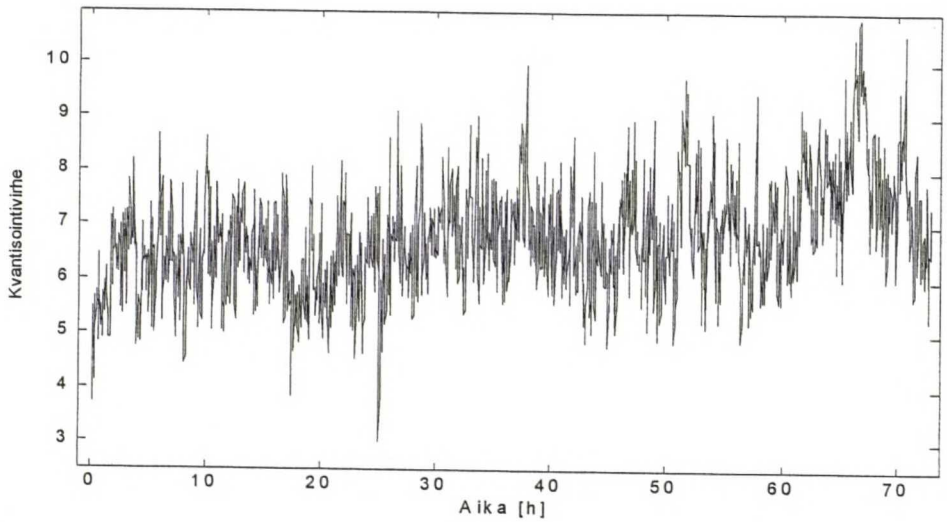
Koko prosessi mallinnettiin klusterianalyysin avulla käyttämällä luvussa 4 esitettyä algoritmia. Opetusdatassa on 500 näytettä 52:sta signaalista. Klustereiden lukumääräksi valittiin 100. Kaikki signaalit normalisoitiin ennen klusterointia.

Klusteroinnin sijoittamat koodivektorit kuvaavat prosessin normaalitoimintaa, joten prosessissa tapahtuva muutos pitäisi näkyä kvantisointivirheen kasvuna.

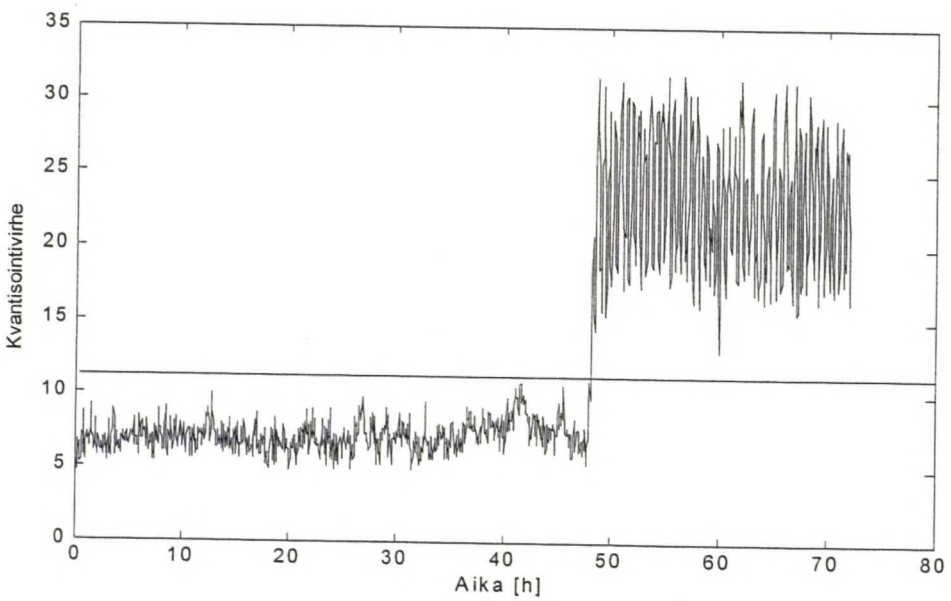
Saatu klusterimalli validoitiin erillisellä validointidatalla vertaamalla opetus- ja validointidatalle laskettua kvantisointivirhettä, josta ei pitäisi pystyä erottamaan kohtaa, jossa data vaihtuu. Validointidata normalisoitiin käyttämällä opetusdatasta laskettuja parametreja. Opetus- ja validointidatalle laskettu kvantisointivirhe on esitetty kuvassa 24.

Seuraavaksi lasketaan klusterimallin kvantisointivirhe datalle, jossa vika näkyy. Kvantisointivirhe on on esitetty kuvassa 25. Venttiili vikaantuu 45:n tunnin kohdalla.

Näytteet kyseiseen hetkeen asti kuvaavat prosessin normaalia toimintaa. Kvantisointivirheen avulla voidaan siis havaita muutos monimutkaisen systeemin käyttäytymisessä.



Kuva 24 Normalisoitu kvantisointivirhe opetus- ja validointidatalle. Validointidata alkaa hetkellä t=25h. Koska opetus- ja validointidatasta lasketun kvantisointivirheen välillä ei ole merkittävää eroa, malli kuvaa dataa suhteellisen hyvin.

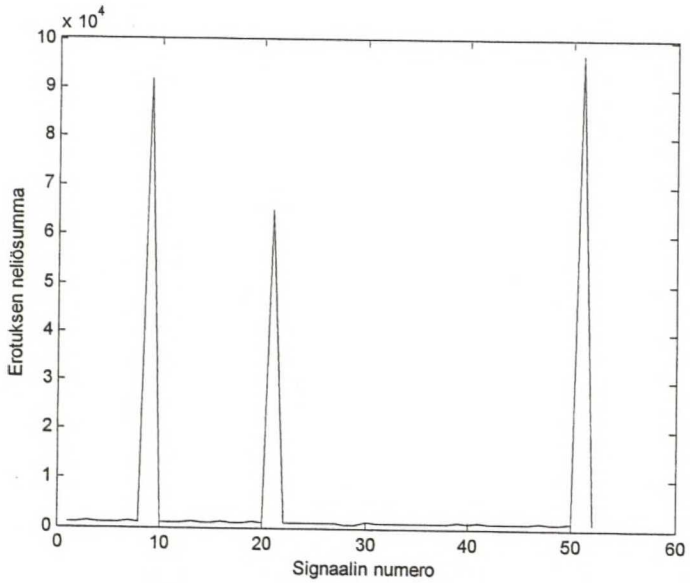


Kuva 25 Kuluneen venttiilin havaitseminen kvantisointivirheen avulla. Klusterianalyysi tehtiin prosessin kaikille 52:lle signaalille, jonka jälkeen koodikirjan avulla voidaan havaita viallisen venttiilin aiheuttama muutos prosessissa. Vikaantuminen tapahtuu hetkellä t=47 tuntia. Kvantisointivirheen raja-arvo on annettu validointidatalle saadun kvantisointivirheen perusteella.

Opetusdataa ja vikatilanteesta kerättyä dataa voidaan verrata tässä kyseisessä tapauksessa toisiinsa laskemalla kaikille signaaleille vikatilanteen ja normaalitilanteen erotuksen neliösumma kaavalla 5.2. Tällä tavalla vikaa kuvaavasta datasta voidaan paljastaa ne signaalit, jotka eroavat eniten normaalitilannetta kuvaavasta datasta (kuva 26).

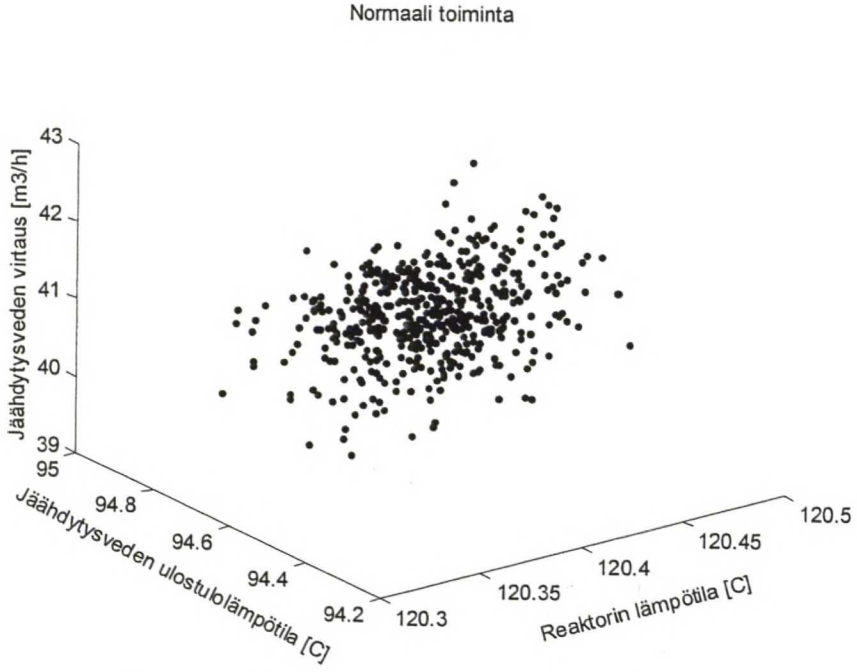
$$d(j) = \sum_{i=1}^{n} (x_n(i,j) - x_v(i,j))^2 \; \forall j = 1..p \tag{5.2}$$

jossa x_n on normalisoitu opetusdata, x_v on normalisoitu vikatilanteesta kerätty data, n on näytteiden lukumäärä ja p on signaalien lukumäärä. Aikasarjojen x_n ja x_v täytyy olla yhtä pitkiä.

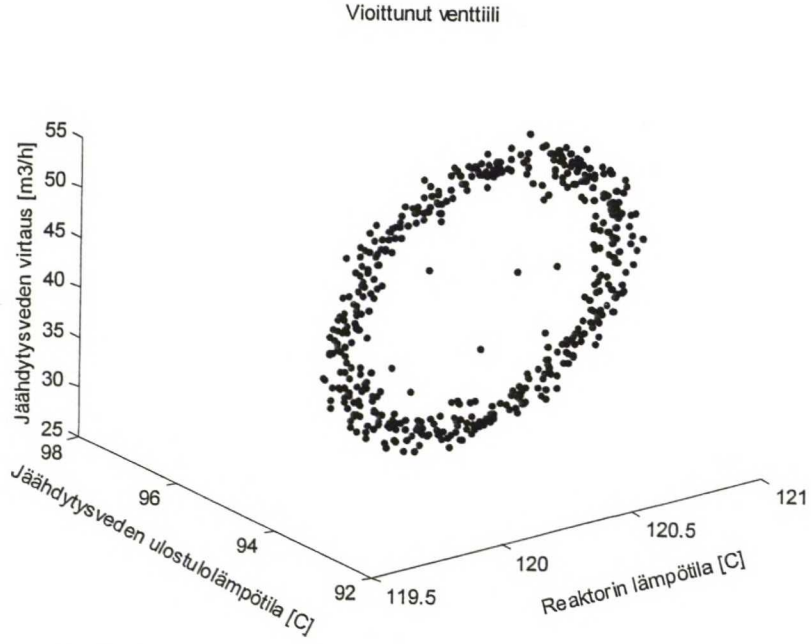


Kuva 26 Normaalin ja vikaantuneen datan erotuksen neliösumma eri signaaleille. Vika vaikuttaa ainoastaan kolmeen signaaliin 52:sta.

Nähdään, että reaktorin jäähdytysveden venttiilissä esiintyvä kitka vaikuttaa ainoastaan kolmeen signaaliin 52:sta. Signaalit ovat reaktorin lämpötila (9), reaktorin jäähdytysveden ulostulolämpötila (21) ja reaktorin jäähdytysveden virtaus (51). Venttiilin vikaantuminen aiheuttaa näihin signaaleihin epänormaalia värähtelyä, joka näkyy muutoksena jäähdytysjärjestelmän ominaiskuviossa. Signaaliarvojen muodostama normaali ja viallinen 'ominaiskuvio' on esitetty kuvissa 27 ja 28.



Kuva 27 Reaktorin jäähdytykseen liittyvä normaalitilanteesta kerätty data. Normaalitilanteessa vaihtelu on satunnaista kohinaa.



Kuva 28 Reaktorin jäähdykseen liittyvä data kun venttiili on vikaantunut. Useassa signaalissa esiintyvä värähtely muodostaa signaaliavaruuteen ellipsin, jolloin värähtely havaitaan kvantisointivirheessä.

Simulaattoriin ohjelmoidusta 21 viasta 13 vikaa aiheutti kvantisointivirheen nousun yli 1.5 kertaiseksi validointidatalle saatuun maksimiarvoon verrattuna ja 3 muuta vikaa aiheutti pienemmän muutoksen.

Tällainen vikadiagostiikkajärjestelmä ottaa huomioon eri ajotilanteet, koska normaali ominaiskuvio voi olla millainen tahansa. Myös vian aiheuttaja voidaan selvittää koodikirjan avulla. Piirrevektorista voidaan etsiä alkiot, jotka eroavat eniten vastaavasta koodivektorista. Tällä tavalla saadaan selville epänormaalit signaalit, joiden tuntemisesta on hyötyä epänormaalin käyttäytymisen syyn selvittämisessä.

Kvantisointivirheen kasvaessa tilastollisesti merkittävästi normaaliarvostaan järjestelmä antaa tästä ilmoituksen, jolloin operaattori voi mahdollisesti korjata tilanteen ajoissa ennen kuin ongelma vaikuttaa laatusuureisiin.

6 Testaus

6.1 Yleistä

Työssä kehitettyä ajotilanteen tunnistusjärjestelmää testattiin UPM-Kymmenen Kajaanin tehtaille asennetussa tietokannassa. Testauksen tarkoitus oli varmistaa työssä kehitetyn laskentalohkon toiminta todellisessa ympäristössä. Testaus jakautui kahteen osaan siten, että ensimmäisessä testissä kokeiltiin klusterointilohkon toimintaa ja tarkkailtiin itseorganisoitumista eräässä massalinjassa.

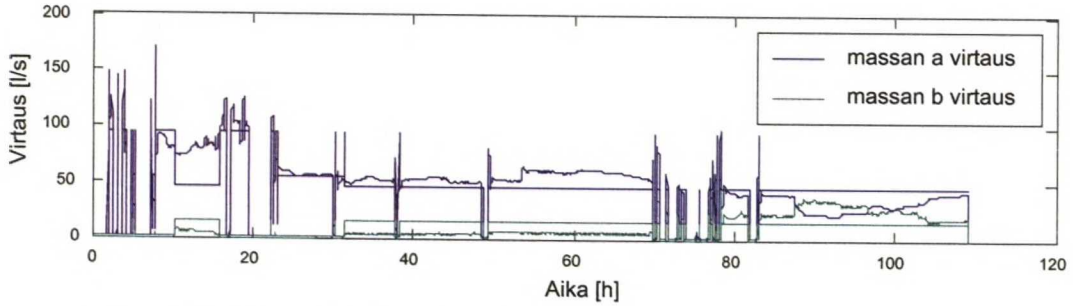
Toisessa testissä laskennan tarkoituksena oli päätellä koneella ajettavan paperilajin tyyppi tietokantaan talletettavien mittausten perusteella. Tietokantaan talletetun lajitiedon perusteella luotiin LBhistory-ohjelmalla lajikohtaisia suorituskykyraportteja säätöpiirien suorituskyvyn lajiriippuvuuden selvittämiseksi.

6.2 Laskentalohkon testaus

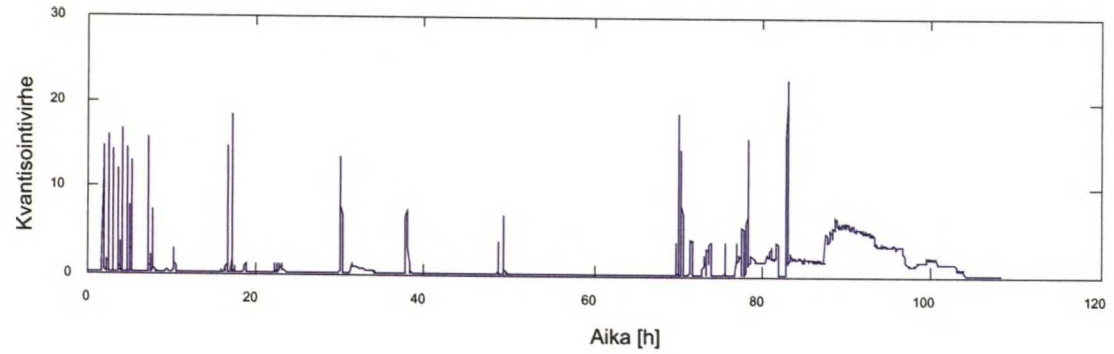
Laskentalohkoa testattiin konfiguroimalla tulosignaaleiksi erään massalinjan kaksi virtausmittausta ja linjassa olevien pumppujen käyntitiedot. Laskentalohkon piti sijoittaa näiden muuttujien muodostamaan avaruuteen 14 koodivektoria. Koodivektoreiden sijainnit tarkistettiin neljän päivän välein kolme kertaa.

Itseorganisoitumisen kehittymistä voidaan arvioida muodostamalla prosessidata uudelleen konfiguraatiotietokantaan talletettujen koodivektoreiden ja DNAhistorian-tietokantaan talletetun ajotilanteen koodinumeron avulla ja vertaamalla tätä tietokannasta luettuun prosessidataan. Kehitys nähdään myös kvantisointivirheen pienentymisenä.

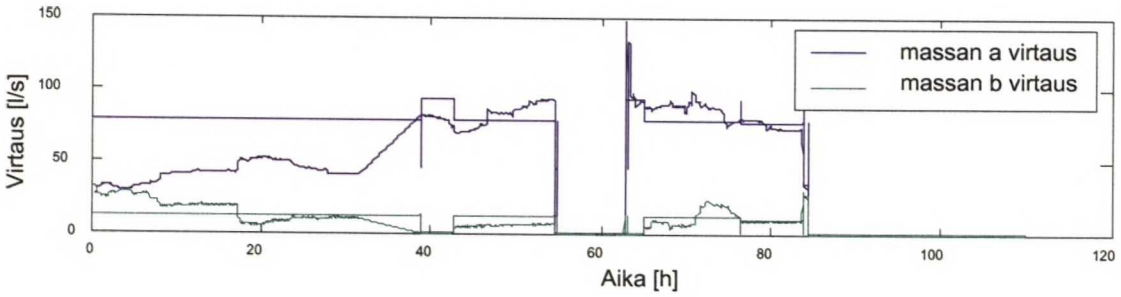
Seuraavista kuvista nähdään, että koodivektorit hakeutuvat näytevektoreiden läheisyyteen. Kuvassa 29 nähdään, että koko data kuvataan ainoastaan muutamalla koodivektorilla, jolloin kuvassa 30 esitetty kvantisointivirhe on myös suuri. Neljä vuorokautta myöhemmin (kuvat 31 ja 32) tilanne on edelleen melkein sama. Kolmannen jakson aikana (kuvat 33 ja 34) laskentalohko on suorittanut algoritmin vaiheen 6, jolloin huonon alkuarvon saaneet koodivektorit on siirretty sinne mistä dataa on löytynyt. Koodivektoreiden avulla muodostettu data seuraa jo melko hyvin tietokannasta luettuja mittauksia. Myös kvantisointivirhe on pienentynyt. Testi osoittaa, että tietokannassa rekursiivisesti suoritettava algoritmi toimii halutulla tavalla.



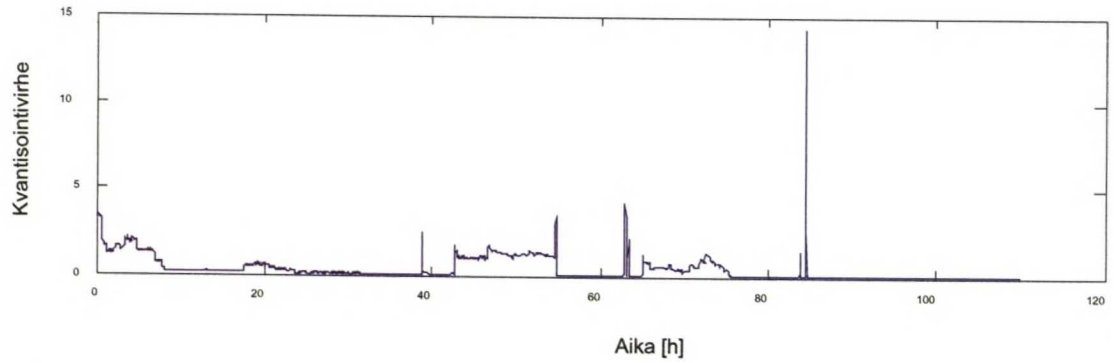
Kuva 29 Prosessista mitattu data ja vastaava koodivektoreiden avulla muodostettu data, kun aikaa on kulunut neljä vuorokautta laskennan aloituksesta.



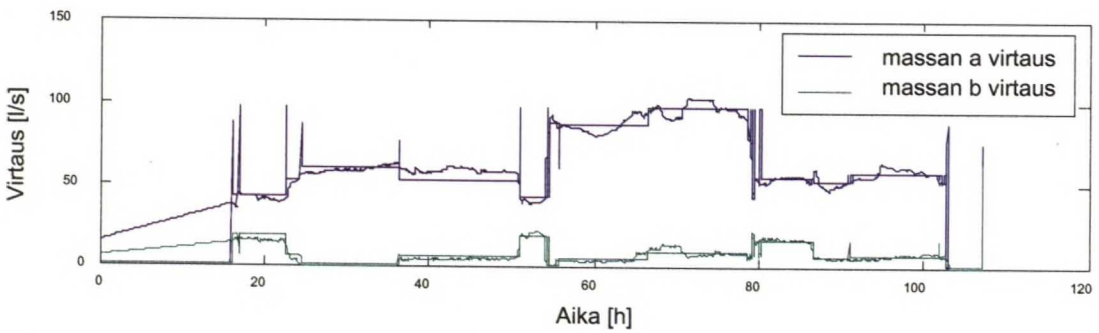
Kuva 30 DNAhistorian-tietokantaan talletettu kvantisointivirhe hetkellä t=0-4 vuorokautta laskennan aloittamisesta.



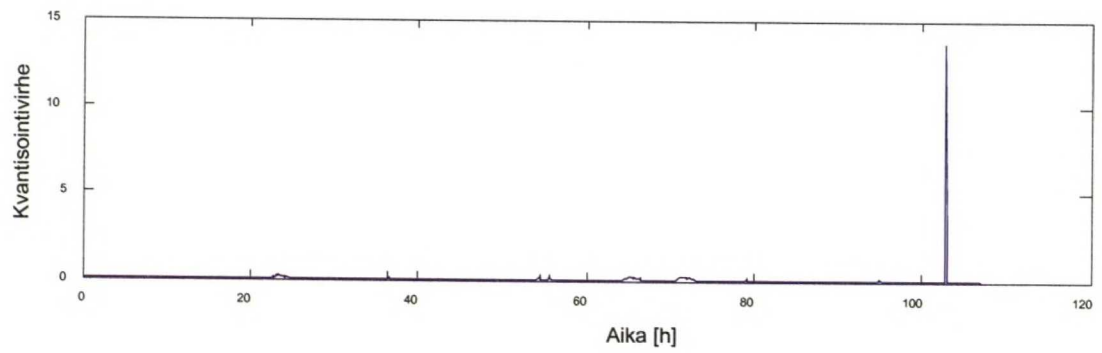
Kuva 31 Prosessista mitattu data ja vastaava koodivektoreiden avulla muodostettu data, kun aikaa on kulunut kahdeksan vuorokautta laskennan aloituksesta.



Kuva 32 DNAhistorian-tietokantaan talletettu kvantisointivirhe hetkellä *t*=4-8 vuorokautta laskennan aloittamisesta.



Kuva 33 Prosessista mitattu data ja vastaava koodivektoreiden avulla muodostettu data, kun aikaa on kulunut 12 vuorokautta laskennan aloituksesta.



Kuva 34 DNAhistorian-tietokantaan talletettu kvantisointivirhe hetkellä *t*=8-12 vuorokautta laskennan aloittamisesta.

6.3 LBhistory

LBhistory helppokäyttöinen työkalu säätöpiirien systemaattiseen suorituskyvyn offline-seurantaan ja raportointiin. LBhistory on osa LoopBrowser tuotetta ja säädön suorituskyvyn evaluointi perustuu samoihin suorituskykyindekseihin.

LBhistory tekee HTML-pohjaisen suorituskykyraportin halutuille säätöpiireille tietyltä ajanjaksolta offline-analyysinä. Tietyllä hetkellä säätöpiirin suorituskyky päätellään sumean logiikan avulla DNAhistorian-tietokantaan talletetuista suorituskykyindekseistä. Päättelyn tuloksena syntyy säätöpiirissä voimassa oleva tilanne. Mahdolliset tilanteet jakautuvat hyviin ja huonoihin tilanteisiin ja niitä vastaavat ilmoitukset on lueteltu taulukoissa 1 ja 2.

Suorituskykyanalyysin tulos esitetään raportissa piirakkadiagrammina, josta nähdään hyvien ja huonojen tilanteiden osuus tutkitusta ajanjaksosta.

Taulukko 2 Huonon suorituskyvyn ilmoitukset.

No	Message
1	Setpoint traces measurement
2	Load disturbance
3	Disturbance peak and measurement noise
4	Noisy load disturbance
5	Control saturated
6	Grade change / load disturbance
7	Process upset
8	Measurement noise
9	Oscillating loop

Taulukko 3 Hyvän suorituskyvyn ilmoitukset.

No.	Message
1	OK / Loop operating normally
2	OK / Control travel extra small
3	OK / Control error extra small
4	OK / Oscillation echo
5	OK / Good control quality, noisy process
6	OK / Good control quality, disturbance peak
7	OK / Good control quality
8	OK / Good control quality, control at rest
9	OK / Variability extra small
10	OK / Control travel and variability small
11	OK / Control error and variability small
12	OK / Control error and control travel small

Muiden analyysien tulokset ovat erilaisia kuvaajia, trendejä, lokeja ja taulukoita. Raportin loppuun tulee yhteenveto, jossa samalla sivulla esitetään kaikkien analysoitujen säätöpiirien suorituskyky sekä näistä laskettu kokonaissuorituskyky. Näistä saadaan jonkinlainen kuva seurattavaan yksikköprosessiin kuuluvien säätöpiirien yleisestä suorituskyvystä.

Raportti voidaan konfiguroida tehtäväksi automaattisesti määrävälein, jolloin säätöpiirien suorituskyvyn kehittymisestä saadaan dokumentoitua tietoa. Mikäli raportti konfiguroidaan siten, että siihen kuuluu vain saman yksikköprosessin säätöpiirejä, voidaan ohjelmalla periaatteessa analysoida myös yksikköprosessin suorituskykyä [Met02a].

6.4 Ajettavan paperin lajityypin tunnistus

Tutkitulla paperikoneella tuotetaan korkealuokkaista SC-painopaperia. Koneella ajetaan kahta eri lajityyppiä, jotka jakautuvat syväpaino- ja offset-laatuihin. Lisäksi molemmista lajityypeistä tuotetaan erittäin vaaleaa versiota. Yhteensä eri lajeja on siis kahdeksan, joita kaikkia voidaan tuottaa eri neliöpainoilla ja eri nopeuksilla.

Opetusvaihe suoritetaan Matlab-ympäristössä DNAhistorian-tietokannasta haetulle opetusdatalle. Opetusvaiheen tarkoituksena on löytää lajityypistä riippuvat signaalit ja tuottaa klusterointimenetelmällä tunnistamisessa tarvittava koodikirja, joka voidaan

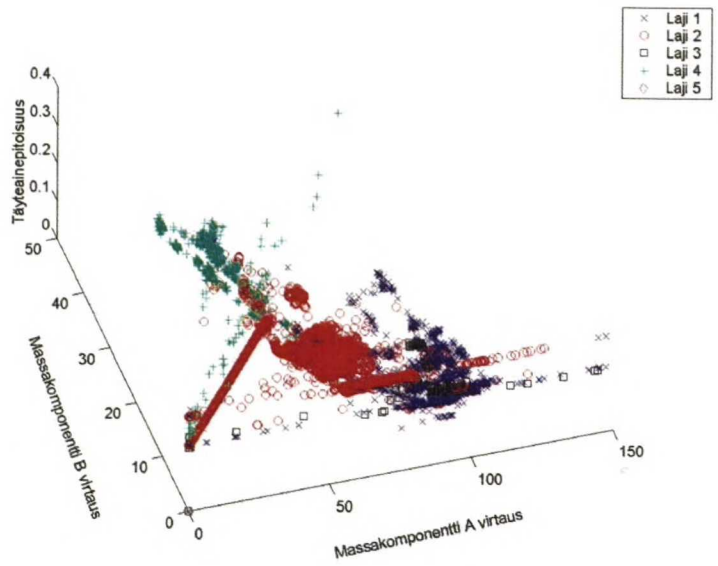
siirtää online-algoritmin käyttöön. On-line algoritmin tehtävä on ainoastaan ylläpitää koodikirjaa ja sopeutua hitaisiin muutoksiin.

Asennetussa tietokannassa on käytettävissä 616 analogiasignaalia ja 170 binäärisignaalia. Lajintunnistukseen ei voida käyttää näitä kaikkia, joten signaalien joukosta täytyy etsiä ne, jotka riippuvat selvästi ajettavasta lajista. Tietokannasta haettiin kahden kuukauden ajalta dataa 14 eri mittaussignaalista, joiden oletettiin riippuvan ajettavasta paperilajista [VTT02]. Näyteväli oli yksi minuutti. Signaalit olivat eri massakomponenttien virtaus- ja sakeusmittauksia, konesäiliöön syötettävän massan kokonaisvirtaus ja sakeus, neliöpainomittaus, täyteainemittaus, kosteusmittaus ja perälaatikon paine. Lisäksi tehtaalta saatiin tieto ajetuista lajeista vastaavalta ajalta.

Usean yrityksen ja erehdyksen jälkeen tunnistukseen valittiin kahden massakomponentin virtausmittaukset ja täyteainepitoisuus. Tietokantaan tallennetaan täyteaineneliömassan absoluuttiarvoa, joten täyteainepitoisuus piti laskea erikseen. Prosessidatassa on lähes poikkeuksetta mukana nollia, joten täyteainepitoisuuden laskennassa käytetään kaavaa 6.1 nollalla jaon estämiseksi. Näiden muuttujien riippuvuus paperilajista on esitetty kuvassa 35.

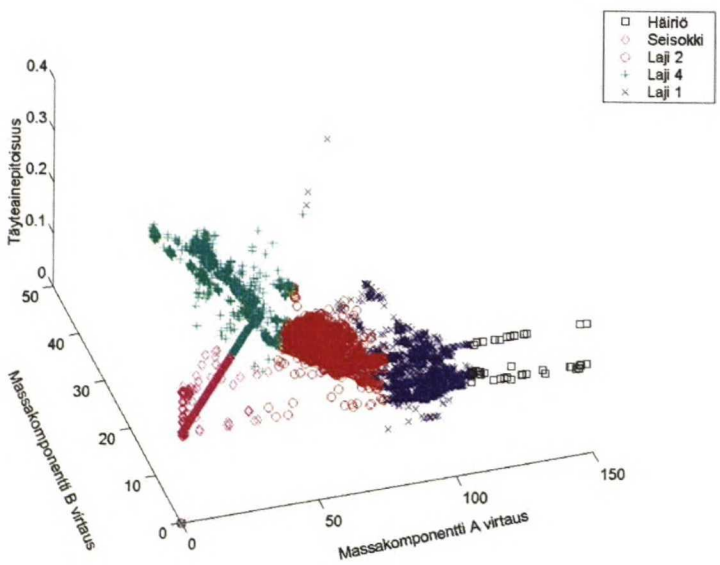
$$x^* = \frac{x_1}{x_1 + x_2} \tag{6.1}$$

, jossa x^* on täyteainepitoisuutta kuvaava tunnusluku, x_1 on täyteaineen neliömassa ja x_2 on kokonaisneliömassa.



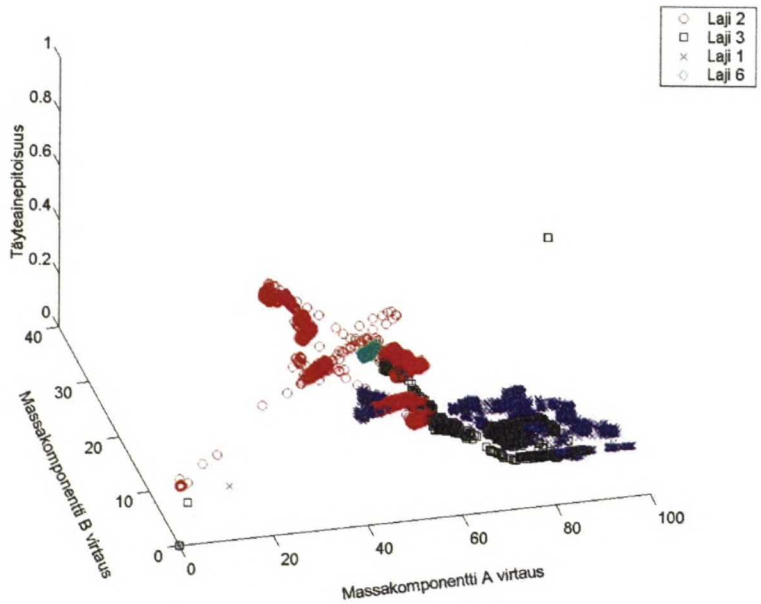
Kuva 35 Tietokannasta haettu data yhdistettynä tehtaalta kysyttyyn lajitietoon. Väri kuvaa ajettua lajia. Eri lajit erottuvat melko selvästi.

Opetusdata klusteroitiin luvussa 4 esitetyllä algoritmilla Matlab-ympäristössä. Koodivektoreiden lukumäärä on viisi. Data käytiin läpi ainoastaan kerran, joten tämä olisi voitu tehdä myös reaaliaikatietokannassa. Näytteet luokiteltiin klusteroinnin synnyttämän koodikirjan avulla ja tulos on esitetty kuvassa 36.



Kuva 36 Klusteroitu data. Klusterianalyysi jakoi datan ryhmiin, jotka vastaavat paperilajeja, seisokkia ja häiriötilannetta. Ajettavan paperilajin tyyppi voidaan tunnistaa klusterianalyysin tuottamien koodivektorien avulla.

Klusterointi löysi datasta eri lajeja kuvaavat ryhmät, joiden perusteella tunnistus voidaan tehdä. Klusteroinnin tulos on esitetty kuvassa 39. Seuraavaksi tulos validoitiin validointidatalla. Validoinnissa käytetty data haettiin opetuksessa käytetyn datan jälkeiseltä kahden viikon ajalta, jolloin oli ajettu lajeja 1, 2, 3 ja 6. Lajia 6 ei esiinny opetusdatassa. Validointidata ja ajettu lajityyppi on esitetty kuvassa 37.



Kuva 37 Validointidata yhdistettynä tunnettuun lajitietoon. Väri kuvaa ajettua paperilajia.

Opetusdatassa lajit 2 ja 4 erottuvat selvästi. Validointidatassa osa lajiin 2 kuuluvista näytteistä on kuitenkin lajin 4 alueella, joten näiden erottelu ei ilmeisesti ole mahdollista. Lajit 2, 4 ja 6 kuuluvat samaan lajityyppiin A. Lajit 1 ja 3 kuuluvat lajityyppiin B. Lajityypit erottuvat selkeästi myös validointidatassa.

6.5 Tunnistusjärjestelmän käyttöönotto

Opetusvaiheen tuloksena syntyi tietämys lajin kannalta oleellisista signaaleista ja lajityypin tunnistuksen perustana toimiva viidestä ajotilanteesta koostuva koodikirja. Nämä siirrettiin ajotilanteen tunnistusjärjestelmän konfiguraatiotietokantaan luvussa 4 esitetyllä tavalla käyttäen tätä varten toteutettuja funktioita.

Konfiguraatiotietokanta ja laskentafunktiot asennettiin jo ennestään asennettuun DNAhistorian-tietokantaan, jonka jälkeen laskenta alkoi pyöriä ja tuottaa tietoa ajettavasta lajityypistä.

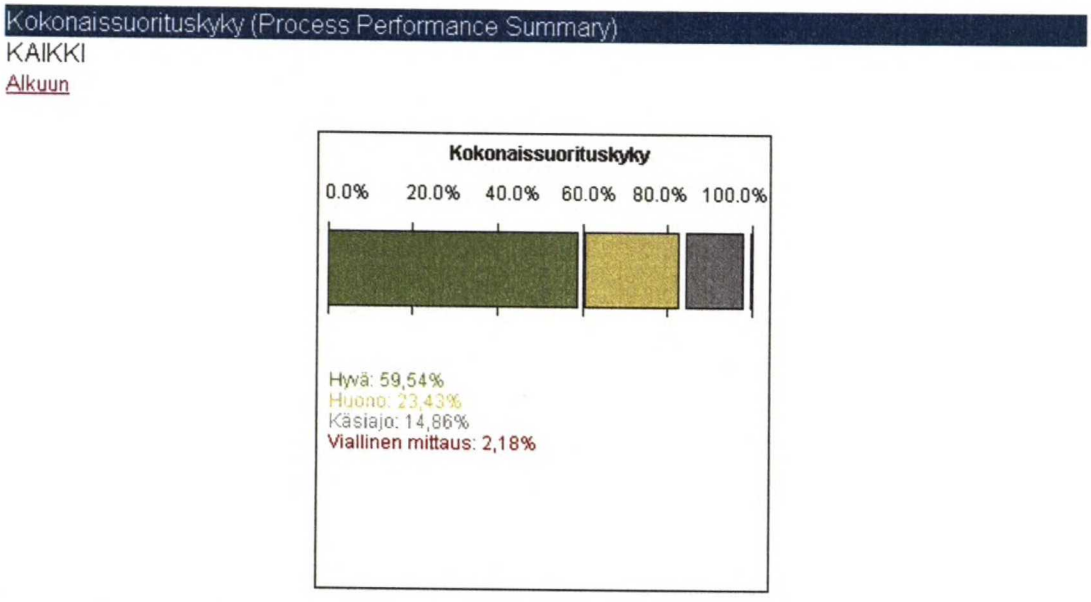
6.6 Ajotilanteen merkitys

LBhistory:ssä datan hakuun voidaan liittää kriteerejä, jolloin dataa haetaan vain niiltä ajanjaksoilta, joilla haun kriteerit täyttyvät. Ohjelmalla luotiin analysoitavasta paperikoneesta lajikohtaisia suorituskykyraportteja yhdeksälle sakeussäätöpiirille. Haun kriteerinä käytettiin klusteroinnin avulla pääteltyä ajotilannetta eli ajettavaa lajityyppiä.

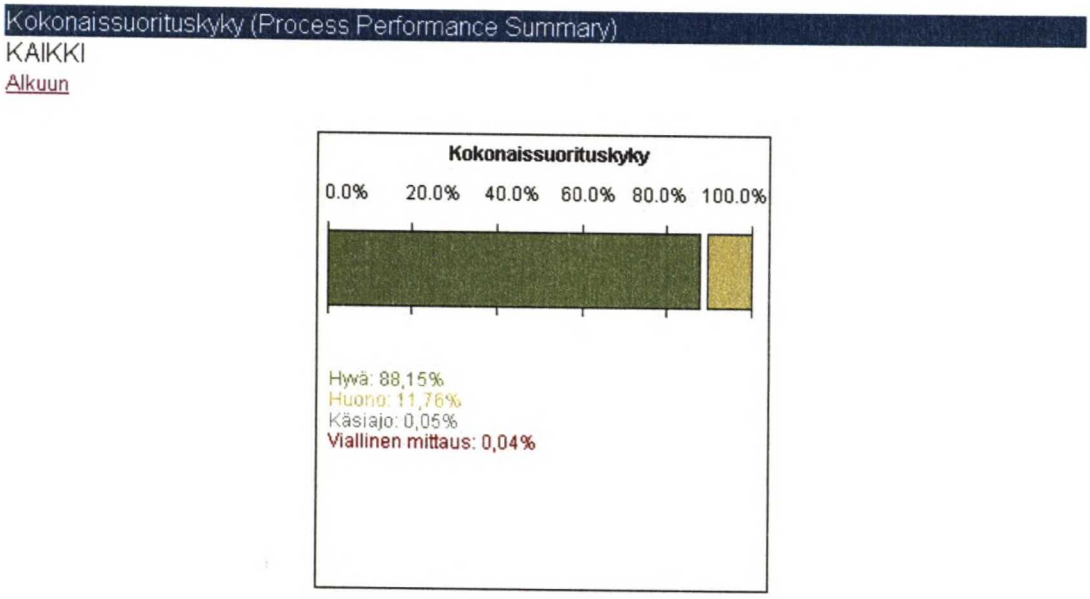
Kaikissa raporteissa käytetty data on peräisin samalta viikolta, jolloin lajityyppiä A ajettiin yhteensä 2 vuorokautta, lajityyppiä B ajettiin 2,5 vuorokautta. Lisäksi paperikone oli seisokissa 3,5 vuorokautta.

Lajikohtaisessa raportissa suorituskykyä on siis analysoitu ainoastaan niiltä ajanjaksoilta, jolloin tätä tiettyä lajityyppiä on laskennan perusteella ajettu. Lisäksi tehtiin raportteja, jossa ajotilannetta ei huomioida. Vertaamalla lajikohtaisia raportteja tähän 'yleisraporttiin', voidaan todeta ajotilanteen merkitys suorituskyvyn määrityksessä.

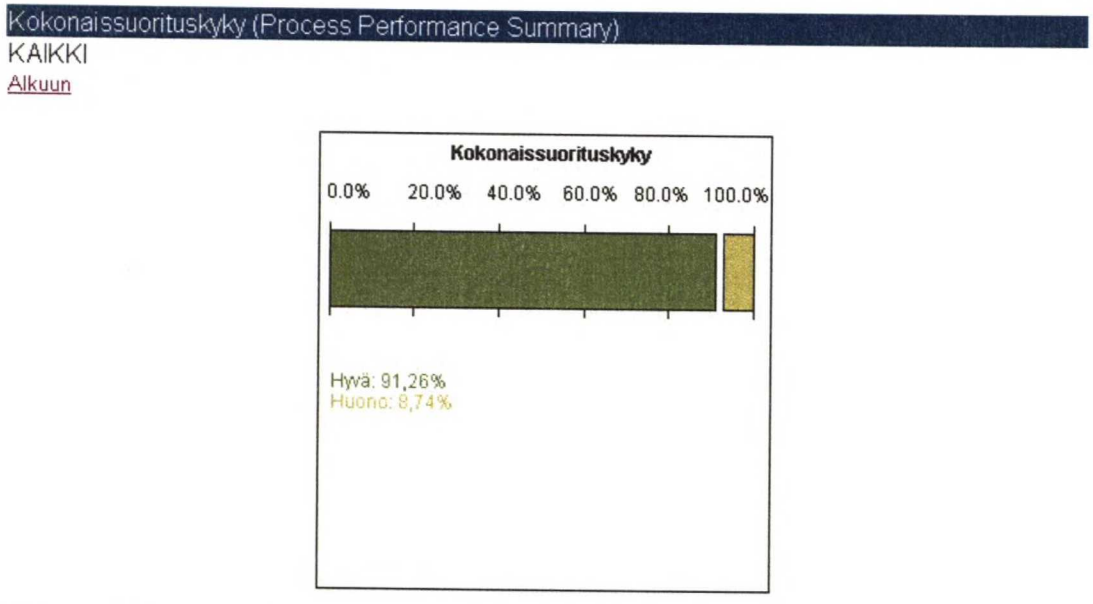
Analyysin tulokset on esitetty kuvissa 38, 39 ja 40. Tilanpuutteen takia raporteista esitetään ainoastaan analysoitujen säätöpiirien kokonaissuorituskyky.



Kuva 38 LBhistory:n avulla muodostettu sakeussäätöpiirien kokonaissuorituskyky, kun ajotilannetietoa ei ole käytettävissä.



Kuva 39 LBhistory:n avulla muodostettu sakeusäätöpiirien kokonaissuorituskyky lajityypille A.



Kuva 40 LBhistory:n avulla muodostettu sakeusäätöpiirien kokonaissuorituskyky lajityypille B.

Mikäli ajotilannetietoa ei ole käytettävissä, analyysin tulos on todellista huonompi. Hyvää suorituskykyä on ainoastaan 59 prosenttia tutkitusta ajasta. Tällainen tulos viittaisi siihen, että sakeussäätimet toimivat 41 prosenttia ajasta huonosti. Tämä ei kuitenkaan pidä paikkansa.

Tyypillinen tulosta vääristävä tapaus on ohjauksen saturoituminen seisokkitilanteessa.

Säätö jää seisokin ajaksi päälle, mutta sen asetusarvoa ei muuteta. Mittausarvo putoaa alarajalle, jolloin säädin yrittää korjata tilanteen ajamalla toimilaitteen tämän rajalle.

Säätöpiirin suorituskyvyn laskenta perustuu näistä signaaleista laskettavien suorituskykyindeksien tulkintaan. Päättelyn tulos huonon suorituskyvyn syyksi on odotetusti "Ohjaus rajalla", koska päättelyyn ei ole käytettävissä informaatiota voimassa olevasta seisokkitilanteesta.

Suodattamalla erikoistilanteet pois saadaan huomattavasti lähempänä totuutta oleva tulos, hyvää suorituskykyä noin 90% ajasta. Huono suorituskyky aiheutui eräästä sakeussäätöpiiristä, jonka läpi virtaa massaa vain osan ajasta. Jos tämänkin säätöpiirin ajotilanne olisi huomioitu, sakeussäätöpiirien suorituskykyanalyysissä saatu tulos olisi ollut vieläkin parempi.

Yleisessä tapauksessa suorituskyvyn seurantajärjestelmän täytyy päätellä voimassa oleva ajotilanne mittauksista. Seisokki ei ole ainoa tulokseen vaikuttava tekijä. Vaihtelevat tuotantonopeudet ja eri ajotavat aiheuttavat vastaavia ongelmia.

Vastaavia lajikohtaisia analyysejä tehtiin noin kahden kuukauden ajan viikon välein, mutta eri paperilajeilla ei tässä kokeessa todettu olevan merkittävää vaikutusta analysoitujen säätöpiirien suorituskykyyn. Säätöpiirit tuntuivat toimivan hyvin, mutta raporteista paljastui muutama hylyn käsittelyyn liittyvä värähtelevä säätöpiiri.

Yksinkertainen testi kuitenkin osoittaa, että ohjelmistot toimivat oikein ja niiden avulla voidaan tutkia tällaisia asioita tarkemmin jatkossa. Erikoistilanteiden suodatus helpottaa raporttien tulkintaa, koska käyttäjän ei tarvitse muistella, mitä koneella kyseisellä viikolla tehtiin.

7 Jatkokehitys

7.1 Ajatuksia kenttälaitediagnostiikasta

Tuotantoprosessissa on useita täysin erityyppisiä ja eri valmistajien laitteita. Tällä hetkellä osaan laitteista on saatavana laitevalmistajan laitekohtaisia diagnostiikkasovelluksia, jotka tuottavat laitekohtaisia laitteen kuntoa kuvaavia tunnuslukuja. Ongelman kenttälaitetasolla muodostavat vanhat laitteet sekä eri valmistajien ja eri laitteiden useat erilliset kunnonvalvontajärjestelmät, joiden antamat ilmoitukset eroavat toisistaan.

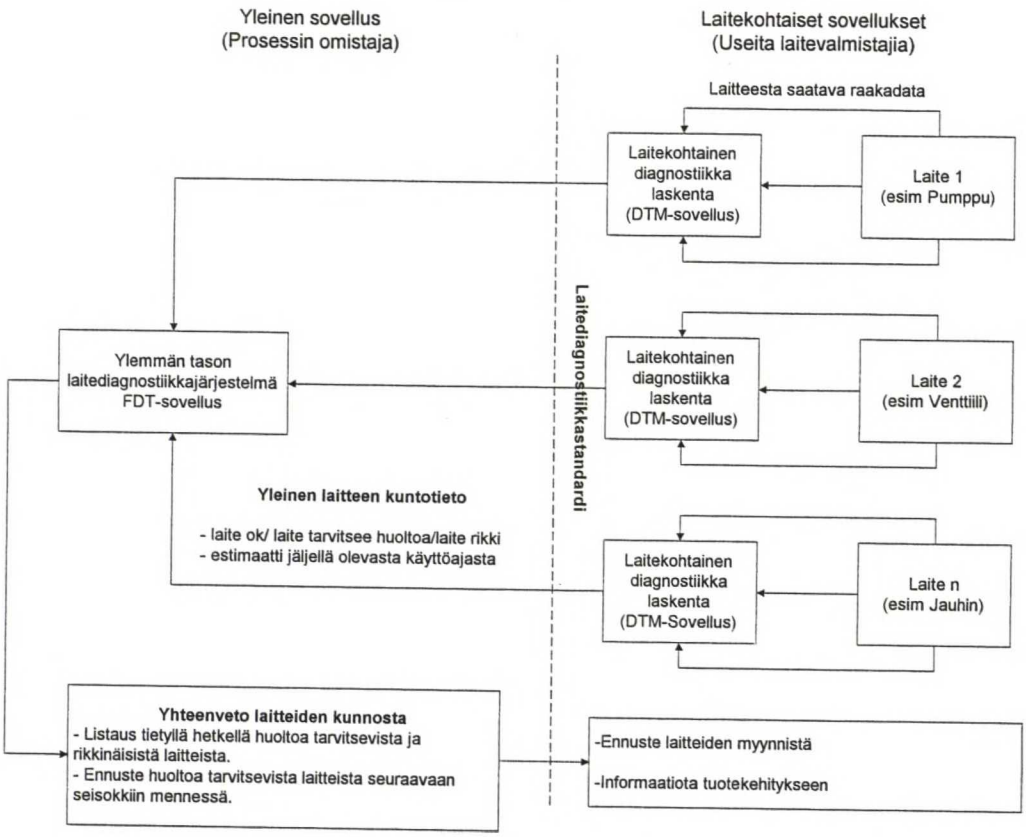
Ongelmat älykkäiden kenttälaitteiden kunnonvalvonnassa eivät siis ole vain teknisiä vaan myös yhteistyön puutteesta johtuvia. Diagnostiikkasovelluksia kehittävien organisaatioiden pitäisi määritellä tarkemmin minkälaista informaatiota kunnonvalvontajärjestelmän on tuotettava. Oikea-aikaisen huollon kannalta olisi tärkeää tietää kestääkö laite seuraavaan huoltoseisokkiin asti vai pitäisikö se vaihtaa. Diagnostiikkajärjestelmät ovatkin tulleet vaiheeseen, jossa yhteistyön lisääminen on välttämättömyys niiden kaupallistamiselle ja kehittämiselle.

Field Device Tool (FDT)-spesifikaatio on ensimmäinen yritys luoda yhteistä rajapintaa ylemmän tason sovellusten ja eri valmistajien älykkäiden kenttälaitteiden välille. Toistaiseksi FDT-spesifikaatio keskittyy lähinnä laitteiden konfigurointiin ja kalibrointiin. Spesifikaatio ottaa kantaa myös laitediagnostiikkaan. Laitekohtaisen diagnostiikan täytyy tuottaa spesifikaation mukainen XML-dokumentti [Fdt01].

Periaatteessa spesifikaation pitäisi ottaa kantaa myös informaation sisältöön. Jos kaikista laitteista saataisiin yhdenmukainen kuntotieto, voitaisiin kehittää tehtaanlaajuinen laitetason kunnossapidon tukijärjestelmä, jonka avulla olisi mahdollista tehdä yksinkertaisia tehtaanlaajuisia yhteenvetoraportteja laitteiden tilasta tietyllä hetkellä. Järjestelmän avulla voitaisiin tehdä myös ennusteita laitteiden tilan kehityksestä. Laitevalmistajan vastuulle jäisi laitekohtaisen diagnostiikkamodulin kehittäminen esimerkiksi DTM-ajuriin.

Periaatteessa laitevalmistajalla on paras tietämys laitteen kulumisesta ja vikaantumisesta sekä näiden oireista, joten on luonnollista, että laitevalmistaja kehittää myös diagnostiikkalaskennan. DTM:n sisällä voidaan käyttää mitä tahansa menetelmiä ja laskea mitä tahansa tunnuslukuja, kunhan laskennan tulos on spesifikaation mukainen.

Tätä on havainnollistettu kuvassa 41. Ongelmaksi jää kuitenkin edelleen se, että laitediagnostiikka tarvitsee informaatiota myös laitteen ympäristöstä. Laitediagnostiikka ei yleensä ole mahdollista pelkän laitteen sisäisen informaation perusteella. Ylimääräiset laitteen ulkopuoliset anturit vaikeuttavat laitteen asentamista ja nostavat kustannuksia.



Kuva 41 Visio laitediagnostiikkajärjestelmästä, joka perustuu standardin mukaisiin laitekohtaisiin laitevalmistajan omiin diagnostiikkamoduleihin. Kaaviossa oletetaan, että diagnostiikkalaskennalla on käytettävissä kaikki tarvittava tieto myös laitteen ympäristöstä.

Älykkäissä laitteissa diagnostiikkamoduuli voidaan toteuttaa laitteen omalla mikroprosessorilla ja diagnostiikan vaatima instrumentointi ja muut vaatimukset voidaan ottaa huomioon jo suunnitteluvaiheessa.

Perinteisten laitteiden osalta tilanne on monimutkaisempi. Online-diagnostiikkaa varten tarvitaan antureita, I/O-kortti ja jonkinlainen laskentaympäristö. Laitteen anturointi jälkeenpäin on kallis ja aikaa vaativa projekti, joten sitä ei kannata tehdä halvimmille laitteille. Laite kuitenkin vaikuttaa ympäristöönsä, joten sen toimintaa voidaan mahdollisesti analysoida prosessimittauksista esimerkiksi reaaliaikatietokannassa tässä työssä esitellyn klusterianalyysin ja kvantisointivirheen avulla.

7.2 Ajatuksia yksikköprosessien monitoroinnista

Yksikköprosessin monitoroinnissa haluttaisiin seurata nimenomaan suorituskykyä, mutta se on vaikea määritellä. Käytännössä kyse on tarpeesta havaita prosessissa ilmeneviä häiriöitä ja muutoksia.

Prosessihäiriöt johtuvat yleensä suunnitteluvirheistä, huonosti viritetyistä säätöpiireistä, huonokuntoisista tai viallisista prosessilaitteista ja antureista tai raaka-aineen laadun vaihteluista. Paperikoneessa tyypillisiä häiriölähteitä ovat esimerkiksi värähtelevät pinnankorkeussäätimet, takertelevat säätöventtiilit ja hylyn käsittely [Nis01]. Suunnitteluvirheistä johtuvien häiriöiden syyn selvittäminen vaatii korkeatasoista prosessiosaamista, joten tällaista diagnostiikkaa on oikeastaan mahdotonta toteuttaa automaattisesti tietokoneohjelmalla.

Usean säätöpiirin muodostamat yksikköprosessit ovat niin monimutkaisia, ettei dynaamisia ilmiöitä ole käytännössä mahdollista mallintaa. Osa häiriöistä voidaan kuitenkin tunnistaa säätöpiirien mittaus- ja ohjaussignaaleista sekä suorituskykyindekseistä muodostetun staattisen mallin avulla. Tällainen malli on eri ajotilanteiden takia epälineaarinen, jolloin sitä ei voida muodostaa perinteisen regressioanalyysin avulla. MLP-tyypisten neuroverkkojen opetus vaatii paljon hyvälaatuista dataa.

Prosessidataa on saatavilla runsaasti, mutta sen laatu on edelleen ongelma. Data on talletettu vaihtuvalla näytevälillä, siinä on yhteyskatkojen aiheuttamia virheitä ja erilaisia prosessin häiriötilanteita. Historiatietokannat pakkaavat dataa siten, että sen käyttö mallinnuksessa voi olla vaikeaa. Myös massiivisten tietomäärien siirto eri sovelluksien välillä aiheuttaa ongelmia.

Rekursiiviset algoritmit tarjoavat ratkaisun näihin ongelmiin. Laskentaa voidaan suorittaa yksittäisille näytteille aina kun näyte todetaan virheettömäksi. Klusterointialgoritmia suorittavalla laskentalohkolla malli muodostuu itsestään, jolloin varsinaista mallinnustyötä ei välttämättä tarvita.

Työssä toteutettu laskentalohko on toteutettu siten, että samaan tietokantapalvelimeen voidaan luoda useita laskentalohkoja. Lohkoja voidaan kytkeä toisiinsa myös hierarkisesti siten, että ylemmälle tasolle välitetään alemman tason kvantisointivirhe, joka kuvaa monitoroitavan prosessin häiriötasoa (kuva 42). Tällä tavalla voidaan monitoroida erittäin monimutkaisia järjestelmiä. Häiriö kulkeutuu hyvin alatasolta ylätasolle, koska sitä ei suodateta keskiarvoistamalla kuten esimerkiksi HALOT-järjestelmässä.

Hierarkian ylätasolla havaittu häiriö voidaan jäljittää hierarkian alatasolle, koska jokaisessa laskentasolmussa häiriön aiheuttava tulosignaali voidaan koodivektorin avulla selvittää. Samalla kaikki laitekohtaiset tunnusluvut muuttuvat hierarkiassa yleiseksi kvantisointivirheeksi, joka kertoo osajärjestelmän häiriötasosta.

Tällä tavalla voitaisiin ratkaista ristiriitainen vaatimus yleiskäyttöisestä ja helposti asennettavasta prosessin diagnostiikkajärjestelmästä. Laskentamoduuleissa olevat mallit

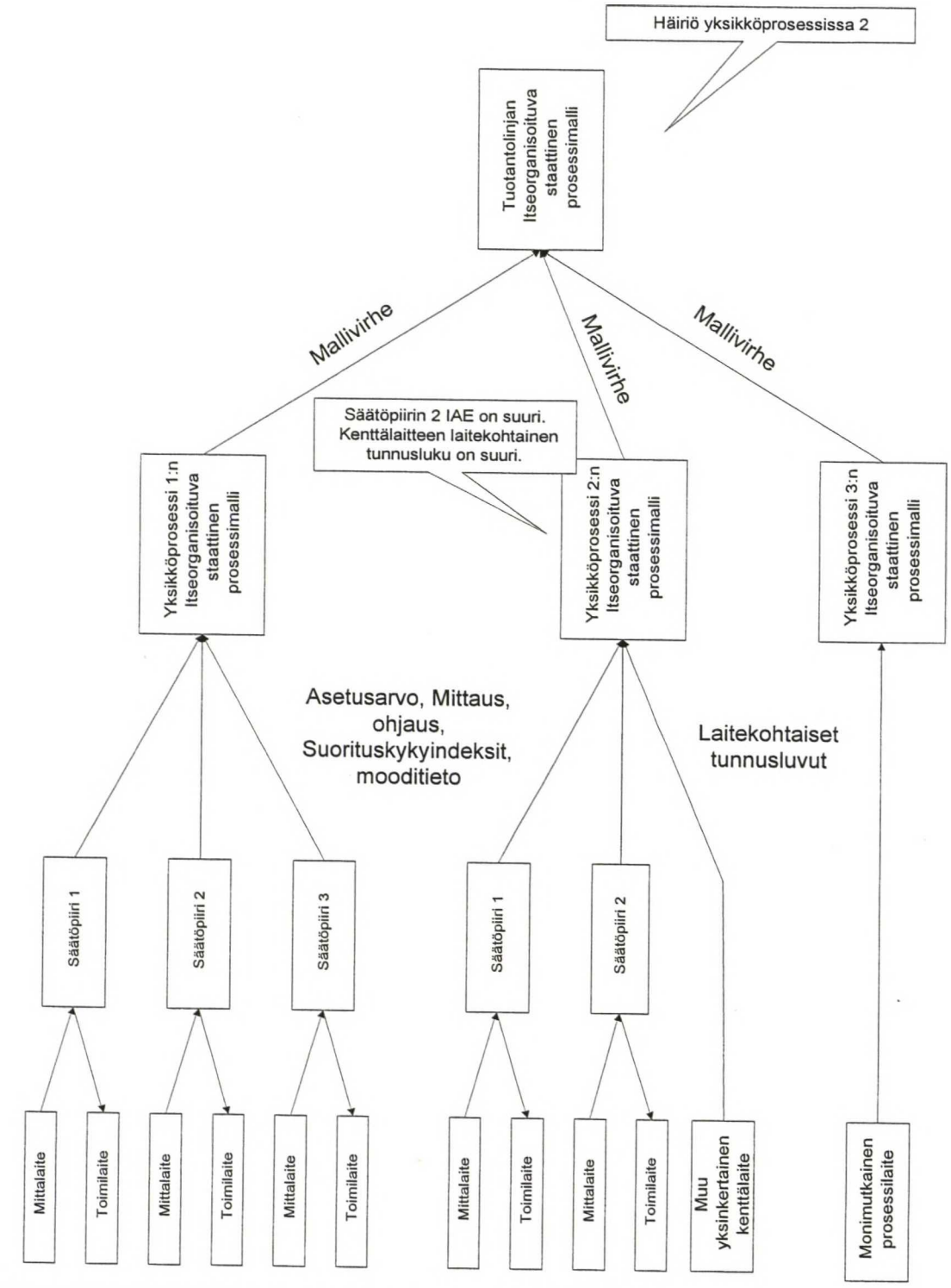
muodostuvat itsestään rekursiivisen algoritmin avulla. Hierarkiatasoja lisäämällä voitaisiin monitoroida aina vaan monimutkaisempia järjestelmiä.

Tällainen rakenne tukisi myös älykkäiden kenttälaitteiden sisäänrakennettua diagnostiikkaa. Mikäli laitediagnostiikka tuottaa laitekohtaisia tunnuslukuja se voitaisiin kytkeä säätöpiiritason laskentasolmuun. Perinteisten kenttälaitteiden osalta joudutaan tyytymään säätöpiiritasolle, koska kenttälaitteesta ei välttämättä saada mitään informaatiota.

Ongelmana tässä on kuitenkin adaptiivisuus. Adaptiivinen järjestelmä sopeutuu muuttuviin olosuhteisiin, joten sellainen ei voi havaita hitaita muutoksia. Toteutetussa klusterointisovelluksessa koodikirjan päivitys voidaan keskeyttää, jolloin havaitaan myös hitaat muutokset. Mahdollisten prosessimuutosten jälkeen se voidaan taas kytkeä päälle, jolloin se sopeutuu muuttuneisiin olosuhteisiin. Käyttäjä ei kuitenkaan tiedä, koska järjestelmä on riittävän "oppinut".

Toinen ongelma on se, että kvantisointivirhe riippuu seurattavien signaalien ja klustereiden lukumäärästä. Järjestelmä kuitenkin olettaa kaikkien signaalien olevan vertailukelpoisia. Alimmalla tasolla signaalien normalisoinnissa skaalausparametrien määrittäminen vielä onnistuu. Seuraavalla tasolla pitäisi normalisoida eri laskentalohkojen kvantisointivirheitä, jotka saattavat saada hyvinkin paljon toisistaan eroavia arvoja. Kvantisointivirhesignaalien tasoja ei kuitenkaan asennusvaiheessa tunneta, joten kvantisointivirhesignaali pitäisi jotenkin standardoida koodikirjan koon perusteella.

Myös signaalien välillä mahdollisesti olevat viiveet vaikeuttavat käyttöönottoa, koska ne pitää ottaa huomioon mallinnuksessa. Mikäli viivettä ei huomioida, signaalien välille ei välttämättä muodostu riippuvuutta eikä mallinnus onnistu.



Kuva 42 Visio hierarkisesta kvantisointiin perustuvasta analyysityökalusta tuotantolinjan tilan analysointiin. Poikkeama yksittäisessä alatason signaalissa havaitaan ylätasolla. Ylätasolla havittu poikkeama voidaan jäljittää hierarkiassa alaspäin lähimmän koodivektorin avulla.

8 Yhteenveto ja johtopäätökset

Työn lähtökohtana ollut analysoitavan järjestelmän eri ajotilanteiden aiheuttama ongelma ratkaistiin klusterianalyysin avulla toteuttamalla tietokantaympäristöön laskentalohko, joka kykenee suorittamaan jatkuvasti rekursiivista klusterointialgoritmia tietokantaan talletettavalle datalle. Eri ajotilanteista saatavat näytevektorit ovat keskenään erilaisia, joten ne voidaan tunnistaa vertaamalla näytevektoria eri ajotilanteita kuvaaviin koodivektoreihin.

Työssä tutustuttiin olemassaoleviin kunnossapidon tukijärjestelmiin, vikadiagnostiikan menetelmiin sekä tunnettuihin klusterointialgoritmeihin. Tietokantatoteutusta varten kehitettiin uusi rekursiivisesti suoritettava klusterointialgoritmi. Työssä pohdittiin klusterianalyysin mahdollisia sovelluksia datan pakkaamisesta vikadiagostiikkaan sekä todettiin, että monet kirjallisuudesta löytyvät itseorganisoituvan kartan sovellukset ovat mahdollisia myös yksinkertaisemmilla klusterointialgoritmeilla. Lopuksi pohdittiin ylemmän tason diagnostiikkasovellusten toteuttamista ja klusterianalyysin hyödyntämistä niissä.

Tämän työn pääpaino on menetelmien suunnittelussa, ohjelmointityössä ja sovellusesimerkeissä. Koodia toteutettiin useille eri alustoille. Varsinainen ydin on kuitenkin yksinkertainen, joten algoritmi voidaan toteuttaa mihin tahansa ympäristöön.

Klusterointialgoritmia suorittava laskentalohko osoittautui yleiskäyttöiseksi. Laskentalohkoa voidaan käyttää sekä analysoitavan datan esikäsittelyyn että analyysien tulosten automaattiseen tulkintaan.

Analyysi voidaan suorittaa esimerkiksi vain tietylle ajotilanteelle. Toinen mahdollisuus on tehdä analyysi, jossa ajotilannetieto huomioidaan esimerkiksi ajotilanteen mukaan muuttuvilla referenssiarvoilla. Mikäli prosessissa on erilaisia ajotapoja, voidaan tutkia myös niiden vaikutusta suorituskykyyn.

Laskentalohkon avulla voidaan muodostaa monitoroitaville muuttujille staattinen malli, jonka avulla voitaisiin havaita poikkeamia kunnonvalvontajärjestelmien tuottamissa sovelluskohtaisissa tunnusluvuissa. Ihmiselle suunnattu raportti luotaisiin vasta kun tunnusluvuissa on jotain poikkeavaa eli kvantisointivirhe on riittävän suuri.

Koodivektoripohjaisesta mallista saatavia etuja prosessin monitoroinnissa MSPC-menetelmiin nähden ovat häiriön havaitseminen ja sen syyn selvittäminen ilman raskaita matriisioperaatioita. Näytettä voidaan verrata lähimpään koodivektoriin, jolloin eniten poikkeavat alkiot ovat syyllisiä kvantisointivirheen kasvuun. Tämän jälkeen syyn selvittäminen on jo huomattavasti helpompaa.

Rekursiivinen opetusalgoritmi mahdollistaa jatkuvan opetuksen ja ratkaisee monia massiivisen datajoukon käsittelyssä ilmeneviä ongelmia. Itseorganisoitumisen avulla voidaan vähentää järjestelmän käyttöönottoon liittyvää konfigurointityötä.

Kehitetty laskentalohko kykenee luokittelemaan näytteitä, mallintamaan ilmiöitä datan perusteella, pakkaamaan dataa tehokkaasti ja sopeutumaan muutoksiin. Ajan kuluessa

laskentalohkon koodikirjaan kerääntyy eräänlaista prosessitietämystä, jota voidaan hyödyntää kunnossapidon tukijärjestelmissä.

Ongelmana itseorganisoituvissa järjestelmissä on se, että ne alkavat toimia vasta jonkin ajan kuluttua asennuksesta kun koodikirja on muodostunut. Käyttäjän on vain luotettava siihen, että järjestelmän toiminta paranee ajan kuluessa. Kvantisointialgorimit mallintavat hyvin vain sen osan mittausavaruudesta, jossa on paljon dataa, joten harvinaiset ajotilanteet aiheuttavat suuremman kvantisointivirheen ja mahdollisesti väärän hälytyksen.

Tietokantayhteyksissä on edelleen parantamisen varaa. Dataa siirretään useiden eri rajapintojen läpi. Datan keruu tietokantaan saattaa katketa esimerkiksi ohjelmistokomponentin kaatumisen takia. Rekursiivisten laskentarutiinien täytyy olla luotettavia etteivät ne kaadu tai hävitä laskennan aikana muodostunutta informaatiota.

Lähdeluettelo

[Air01] Airikka, P. 2001. Control Performance Monitoring Tools. Sisäinen dokumentti.

[Alh00] Alhoniemi, E.,Himberg, J., Parhankangas, J. Vesanto, J. 2000. SOMToolbox for Matlab. Luettavissa <http://www.cis.hut.fi/projects/somtoolbox/>.

[Alh95] Alhoniemi, E. 1995. Monitoring of Complex Processes Using the Self-Organizing Map. Diplomityö. Teknillinen korkeakoulu, Tietotekniikan osasto. Espoo

[Alh99] Alhoniemi, E. Hollmén, J. Simula O., Vesanto, J. 1999. Process Monitoring and Modelling Using the Self-Organizing Map. In Integrated Computer Aided Engineering, IOS Press, Vlo 6. No. 1. p. 3-14.

[Bez92] Bezdek, J.;Hathaway,M.; Sabin, M.;Tucker, W. 1992. Fuzzy Methods for Pattern Recognition, IEEE Press, New York.

[Bot95] Bottou, L.; Bengio, Y. 1995 (viitattu 22.10.2002). Convergence properties of the K-means algorithm. In Advances in Neural Information Processing Systems, volume 7. MIT Press. Luettavissa <http://citeseer.nj.nec.com/bottou95convergence.html>

[Bra01] Braatz, R. 2001.Tennessee Eastman simulated process data. Luettavissa <http://brahms.scs.uiuc.edu/lssrl/software/TE_process/>

[Chi01] Chiang, L.; Russell, E.; Braatz, R. 2001. Fault Setection and Diagnosis in Industrial Systems. London. Springer-Verlag.279 s. ISBN 1-85233-327-8

[Dem77] Dempster, A.; Laird, N.;Rubin, D. 1977. Maximum Likelihood Estimation from Incomplete Data via the EM Algorithm. Journal of The Royal Statistical Society, 39(1) p. 1-38.

[Dow93] Downs, J.; Vogel, E. 1993. A plant-wide industrial process control problem. Computers & Chemical Engineering 17(3), p. 245-255.

[End02] Endén, P. Prosessien monitorointijärjestelmän online-testaus ja tulosten analysointi. 2002. Diplomityö. Teknillinen korkeakoulu, Kemian tekniikan osasto. 108+28 s.

[Fay00] United States Patent 6 012 058. 2000. Scalable system for K-means clustering of large databases. Microsoft Corporation, Redmond, WA. Fayyad; Usama (Mercer Island, WA); Bradley; Paul S. (Madison, WI); Reina; Cory (Kirkland, WA). 042540. March 17, 1998. January 4, 2000.

[Fay01] United States Patent 6 263 337. 2001. Scalable system for expectation maximization clustering of large databases. Microsoft Corporation, Redmond, WA. Fayyad; Usama (Mercer Island, WA); Bradley; Paul S. (Madison, WI); Reina; Cory (Kirkland, WA). 083906. May 22, 1998. July 17, 2001

[Fdt01] Profibus. 2001.Profibus Guideline: FDT Interface Specification v.1.2.

[Fri00] Fridzke, B. 2000. Clustering Methods for Data Mining and Data Analysis. Luettavissa <http://www.ki.inf.tu-dresden.de/~fritzke/lectures/clustering.html>

[Fri01] Friman, M.; Välisuo, M.; Kunnas, A.; Ikonen, K. 2001. Käytännön kokemuksia säädön suorituskyvyn seurannasta. Automaatio 2001 Helsinki.

[Goe01]United States Patent 6 216 066. 2001. System and method for generating alerts through multi-variate data assesment. General Electric Company, Schenectady, NY. Goebel Kai (Balston Lake, NY); Doel David (Maineville, OH). 108 359, July 1, 1998, April 10, 2001

[Gra84] Gray, R. 1984. Vector Quantization. IEEE ASSP Magazine. April 1984. p 4-29.

[Grö00] Grönbärj, M. Teollisuusprosessien vikadiagnostiikkajärjestelmien päättelymekanismit. 2000. Diplomityö. Teknillinen korkeakoulu, Kemian tekniikan osasto. Espoo. 78 s.

[Har97] Harju, T., Marttinen A. 1997. Säätöpiirien analysointimenetelmät. Automaatiopäivät -97. Helsinki. 1997.

[He02] Quiang He.2002. (viitattu 22.10.2002) DBTool -- Another Database Toolbox for MATLAB. Luettavissa <http://go7.163.com/energy/matlab/dbtool.htm>

[Hil02] Hiltunen, J. 2002 .Metsäteollisuuden osaprosessien vikadiagnostiikka. Metsäteollisuuden automaation Mini-Symposium. 21.8.2002. Oulun yliopisto. Oulu. Luettavissa: <http://cc.oulu.fi/~posyswww/ajankohtaista/minisympo/Hiltunen.pdf>

[Hol96] Hollmén, J. 1996. Process Modelling Using the Self-Organizing Map. Diplomityö. Teknillinen Korkeakoulu. Tietotekniikan osasto. Espoo.

[Hyö01] Hyötyniemi, H. 2001. Multivariate regression - Techniques and tools. Helsinki, Finland. Helsinki University of Technology, Control Engineering Laboratory. 207 p. (Helsinki University of Technology, Control Engineering Laboratory Report 125). ISBN 951-22-5587-1

[Jai88] Jain, A. K.; Dubes, R. C. 1988. Algorithms for Clustering Data. Englewood Cliffs, NJ. Prentice Hall. 320 s. ISBN 0-13-022278-X

[Jok91] Jokinen, P. 1991. Continuously Learning Nonlinear Networks with Dynamic Capacity Allocation. Väitöskirja. Tampereen teknillinen korkeakoulu. 100 s. ISBN 951-721-715-3

[Kas92] Kasslin, M. 1992. Itseorganisoituvien piirrekarttojen käyttö kunnonvalvonnassa. Diplomityö. Teknillinen korkeakoulu, Tietotekniikan osasto. Espoo

[Kau90] Kaufman L.; Rousueeuw P.J. 1990. Finding Groups in Data: an Introduction to Cluster Analysis. New York (NY). Wiley. 342 s. ISBN 0-471-87876-6

[Kiv01] Kivikunnas, S. 2001. Ajotilanteen tunnistus. Sisäinen HALOT-projektin dokumentti.

[Kiv02a] Kivikunnas, S. 2002. Halot2-projektin loppuraportti. VTT-elektroniikka. Sisäinen dokumentti.

[Kiv02b] Kivikunnas, S.;Hintikka, J. 2002. Hierarkkinen analysointijärjestelmä osaprosessien tilan tarkkailuun. Metsäteollisuuden automaation Mini-Symposium. 21.8.2002. Oulun yliopisto. Oulu. Luettavissa: <http://cc.oulu.fi/~posyswww/ajankohtaista/minisympo/Kivikunnas.pdf>

[Koh95] Kohonen, T. 1995. Self-Organizing Maps. Berlin. Springer-Verlag. 362 s. ISBN 3-540-58600-8

[Kou02] Kourti, T. 2002. Process Analysis and Abnormal Situation Detection: From Theory To Practice. IEEE Control System Magazine. October 2002. p.10-25.

[Lai00] Laininen, P. 2000. Mat-2.104 Tilastollisen analyysin perusteet kurssin opetusmoniste Kevät 2000. Teknillinen korkeakoulu.

[Lai94] Laine, S.; Jämsä-Jounela, S-L. 1994. On-line determination of the ore type using neural networks and cluster analysis.Minerals Engineering '94, Lake Tahoe, USA. USA, Minerals Engineering '94, SME/AIME.

[Lep02] Leppäkoski, J.; Oksanen, J.; Jaatinen, E. 2002. Kunnossapidom optimointi parantaa laitoksen luotettavuutta ja suorituskykyä. Automaatioväylä. 5/2002. s 14-15.

[Läh02] Lähdemäki, R. Osaprosessin virheellisen toiminnan vaikutus lopputuotteeseen. 2002. Diplomityö. Oulun yliopisto, Prosessitekniikan osasto. Oulu. 90+17+9 s.

[Mar02] Martin,E.; Morris, J.; Lane, S. 2002. Monitoring Process Manufacturing Performance. IEEE Control System Magazine. October 2002. p.26-39.

[Mar93] Martinez, T.M.; Berkovich, S.G.; Schulten, K.J. 1993. "Neural gas"-network for vector quantization and its application to time series prediction. IEEE Transactions on Neural Networks. Vol 4. No 4. p 558-569.

[Mar99] Marttinen, A.;Friman, M. 1999. New Control Performance Measures for Industrial Sub-Processes. Automaatiopäivät 1999.

[McQ67] MacQueen, J. 1967. Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. University of California Press.

[Met01a] Metso Automation. 2001. Neles FieldBrowser™ Solution for Predictive Maintenance. Esite

[Met01b] Metso Automation. 2001. Neles ValvGuard™ Identifies ESD Valve Leakages. Esite

[Met02a] Metso Automation. 2002. LBhistory 2.0 User's Manual

[Met02b] Metso Automation. 2002. LoopBrowser 2.0 User's Manual

[Met02c] Metso Automation. 2002. ND800 Field bus Positioner -Clear choice for the future. Esite

[Met02d] Metso Automation. 2002. Sensodec 6S Unique machinery condition and runnability monitoring. Esite

[Met02e] Metso Automation. 2002. SMART-PULP Sellu- ja paperiprosessien ylivoimainen sakeuslähetin. Esite

[Mic98] Microsoft Corporation. The ODBC Programmer's Reference. Luettavissa <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/odbcabout_this_manual.asp>

[Mil95] Milton, J.; Arnold, J. 1995 Introduction to probability and statistics. Third edition. McGraw-Hill. ISBN-0-07-113535.

[Nis01] Nissinen, A.;Huhtelin, T;,Korpela M. 2001. Märänpään stabiilius laimennusperälaatikkoympäristössä; Säädön haasteet, häiriölähteet ja säätöratkaisut. PIRA 2001 Göteborg runnability conference. Göteborg. Sweden

[Nor00] Nørgaard,M. 2000. "Neural Network Based System Identification Toolbox," Tech. Report. 00-E-891, Department of Automation, Technical University of Denmark. Luettavissa <http://www.iau.dtu.dk/research/control/nnsysid.html>

[Ord00] Ordonez, C.; Cereghini, R. 2000. SQLEM: Fast clustering in SQL using the EM algorithm. In: ACM SIGMOD Conference, pages 559-570. Luettavissa <http://citeseer.nj.nec.com/ordonez00sqlem.html>.

[Par01] Parikka, R.; Ahlroos, T.; Halme, J.; Miettinen, J.; Salmenperä, P.; Lahdelma, S.; Kananen, M.; Kantola, P. 2001. Monitorointi ja diagnostiikka.VTT , Espoo. 55 s. VTT Tiedotteita - 2098. ISBN 951-38-5828-6; 951-38-5829-4

[Pen01] Pennanen, J. 2001. NeuroSense - a tool for process modeling and analysis. Nordic Matlab Conference. October 2001. Oslo Norway.

[Rau02] Rautiainen, H. 2002. Condition Monitoring of a Paper Mill Increases Operational Reliability and Productivity.

[Rin00] Rinta-Runsala, E. 2000. Paper Machine Monitoring Using Self-Organizing Neural Networks. Diplomityö.Teknillinen Korkeakoulu. Teknillisen fysiikan ja matematiikan osasto. Espoo

[Saa02] Saarela, O. 2002. Multivariate Autoregressive Analysis In Locating The Origin of Fluctuation in Continuous Industrial Processes. Väitöskirja. Tampereen teknillinen korkeakoulu. 137 s. ISBN 952-15-0808-6.

[Sql92] ISO/IEC 9075:1992, Information Technology --- Database Languages --- SQL, ANSI X3.135-1992, Database Language SQL

[Str01] Strom, E., Monsen J., Niemelä, I. 2001. Predicting Valve Maintenance Saves Costly Process Downtime A Case Study. In: Pulp & Paper reliability and Maintenance Conference and Exhibition. November 5-9, 2001 Atlanta, Georgia

[Tep99] Teppola, P.; Mujunen S-P.; Minkkinen P. 1999. Adaptive Fuzzy C-Means clustering in process monitoring. Chemometrics and Intelligent Laboratory Systems. Vol 45. No 1-2. p. 23-38

[Ter02] Tervaskanto, M.;Hiltunen, J.;Hietanen, T.; Ahvenlampi, T.; Kortela, U. 2002. Vikadiagnostiikka metsäteollisuuden osaprosessien toiminnan optimoinnissa - loppuraportti. 100 s. ISBN 951-42-6682-X.

[Tun98] Tunkkari, I. 1998. Sähkömoottorien kunnonvalvonta. Seminaarityö. Luettavissa <http://www.ee.lut.fi/lab/sahkomarkkina/kurssit/seminaari/kevat98/moottorien_kunnonvalvonta-Tunkkari.pdf>

[VTT02] VTT Tuotteet ja Tuotanto. 2002. Knowpap 4.0 Paperitekniikan ja automaation oppimisympäristö. Licentia OY.

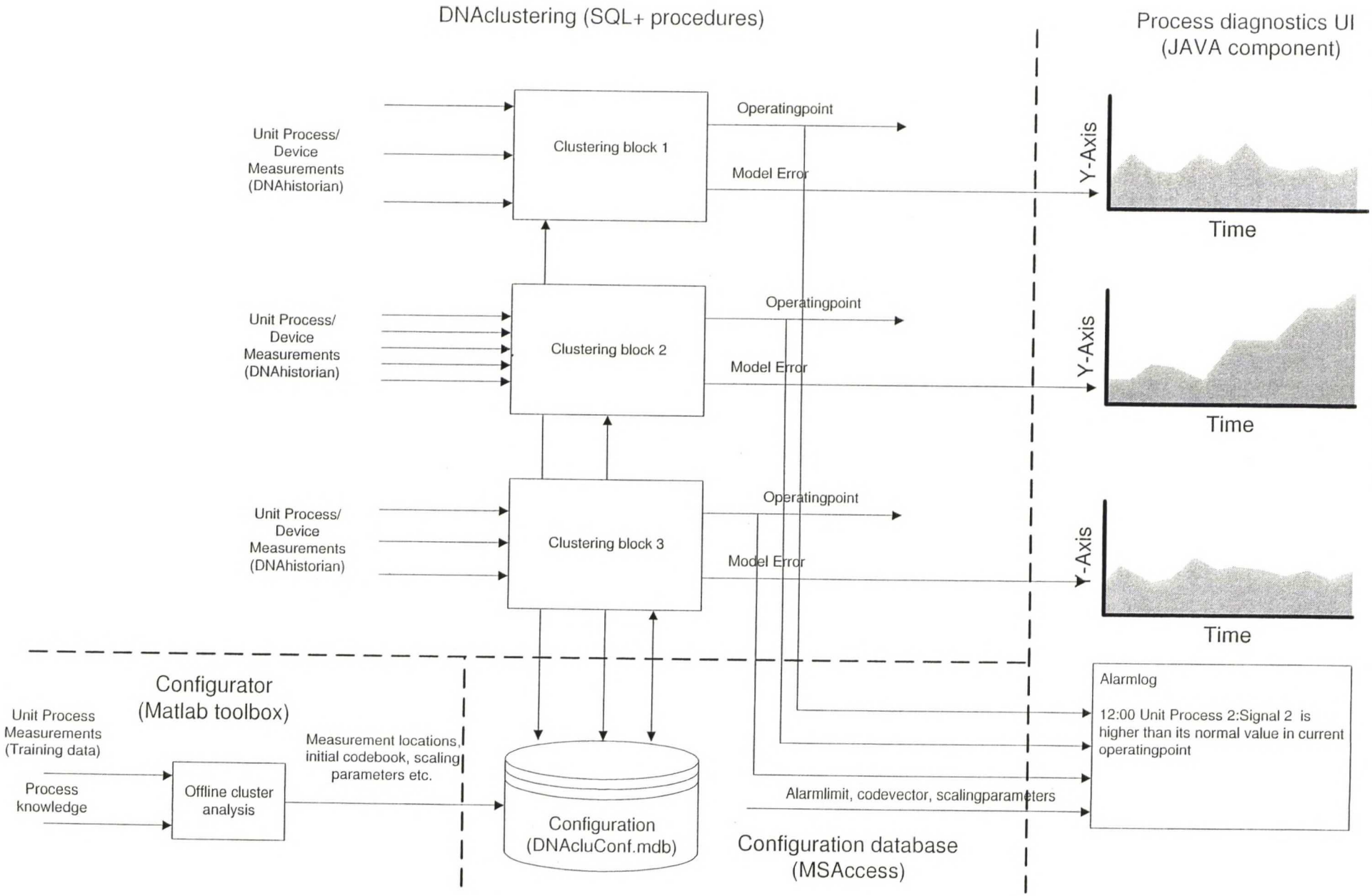
[Yli01] Yli-Petäys, J. 2001. APROS-ohjelmiston hyödyntäminen voimalaitosten simuloinnissa. Diplomityö. Oulun Yliopisto.Prosessi- ja Ympäristötekniikan osasto. Oulu

[Yli94a] Ylinen, R. 1994. Cluster based modelling of processes with unknown qualitative variables. In: Third International Workshop on Computer Aided Systems Theory (EUROCAST '93), Las Palmas, Spain, February 22-26, 1993. Berlin, Heidelberg, Springer-Verlag, s. 268-281.

[Yli94b] Ylinen, R.; Jämsä-Jounela, S-L.; Miettunen, J. 1994. Use of cluster analysis in process control.12th IFAC World Congress, Sydney, Australia, 18-23 July, 1993. Australia, Pergamon, IFAC, Proc. Vol. 4, p. 645-648.

[Ylö99] Ylöstalo, T.; Hyötyniemi, H. 1999 (viitattu 22.10.2002). Model Library Based Adaptive Control - Implementation Aspects.In European Control Conference ECC 99, Karlsruhe, German. Luettavissa <http://saato014.hut.fi/Hyotyniemi/publications/99_ecc.htm>

[Åst95] Åström, Wittenmark. 1995. Adaptive Control. Reading(MA). Addison Wesley. 574 s. ISBN 0-201-55866-1


DNAclustering (SQL+ procedures)
Process diagnostics UI
(JAVA component)
Unit Process/
Device
Measurements
(DNAhistorian)
Clustering block 1
Operatingpoint
Model Error
Y-Axis
Time
Unit Process/
Device
Measurements
(DNAhistorian)
Clustering block 2
Operatingpoint
Model Error
Y-Axis
Time
Unit Process/
Device
Measurements
(DNAhistorian)
Clustering block 3
Operatingpoint
Model Error
Y-Axis
Time
Configurator
(Matlab toolbox)
Unit Process
Measurements
(Training data)
Process
knowledge
Offline cluster
analysis
Measurement locations,
initial codebook, scaling
parameters etc.
Configuration
(DNAcluConf.mdb)
Alarmlog
12:00 Unit Process 2:Signal 2 is
higher than its normal value in current
operatingpoint
Alarmlimit, codevector, scalingparameters
Configuration database
(MSAccess)

Liite 2 Matlab funktiot

```
DNAclustering Configurator.
  Version 1.0 (R12.1)  09-Dec-2002


  Data acquisition.

    clu_get_data                - Get data from DNAhistorian-database.
    clu_dbquery                 - Execute an sqlquery.
    clu_get_clusters            - Read current cluster locations from
                                  configuration database (if exists).
    clu_getvalue                - Zero order hold for timeseries with
                                  different sampling time


  Data mining and training.

    clu_pca                     - PCA analysis.
    clu_lowpassfilter           - Reduce noise from signal.
    clu_sort_by_pc              - Sort multidimensional samples by most
                                  significant principal component.
    clu_num_of_clusters         - Compute quantization errors for
                                  different number of clusters.
    clu_em                      - Expectation Maximization-clustering
                                  algorithm.
    clu_k_means                 - K-means clustering algorithm.
    clu_fcm                     - Fuzzy c-means clustering algorithm.
    clu_set_clusters            - Online clustering algorithm.
    clu_train                   - Perform the cluster analysis using
                                  online clustering algorithm.
    clu_normalize               - Perform autoscaling for given data.
    clu_denormalize             - Perform rescaling for given data.
    clu_sim                     - Compute closest codevectors and
                                  errors.
    clu_classify                - Divide the data to classes using given
                                  codevectors.


  Implementation.

    clu_create_confdb           - Add new clustering block to
                                  configuration database.
    clu_addtodna                - Create new records to DNAhistorian and
                                  start clustering.

Copyright 2002 Metso Field Systems OY.
Author: Ville Hietanen
$Revision: 1.0 $  $Date: 2002/05/12 15:30:00$
```